National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canada

University of Alberta

Analysis of the Conspiracy Numbers Algorithm

by

Lisa Naomi Lister

A thesis

submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Edmonton, Alberta

Spring, 1990

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

ISBN 0-315-60301-1

NL-339 (r. 88/04) c

Canada

micro

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Lisa Naomi Lister

TITLE OF THESIS: Analysis of the Conspiracy Numbers Algorithm

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1990

Permission is hereby granted to University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Signed) Lisa Lister

Permanent Address:

11511-42 Avenue
Edmonton, Alberta
Canada T6J 0W4

Dated Mar. 13, 1990

The University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled *Analysis of the Conspiracy Numbers Algorithm* submitted by **Lisa Lister** in partial fulfillment of the requirements for the degree of Master of Science

.....
Supervisor

.....

.....

.....

Date Feb. 14 1989

Abstract

McAllester's Conspiracy Numbers algorithm is a minimax search procedure which builds game trees to a variable depth without application-dependent knowledge. A lower bound analysis of this algorithm shows that the algorithm expands an exponential number of nodes in a preset pattern before using the specific game tree to determine where the search tree should grow.

A modification of McAllester's Conspiracy Numbers algorithm is presented. Lower bound analysis of the modified algorithm, CNEW, shows that the algorithm expands only a quadratic number of nodes in a predictable manner before using the actual game tree to decide where to expand the search tree. It is thought that the improvement seen in the best case of CNEW will also be apparent in the average case if the two algorithms are compared experimentally.

Acknowledgements

The author would like to take this opportunity to thank the people who helped make this work possible. In particular, I would like to thank my supervisor Jonathan Schaeffer for his guidance throughout this research.

Table of Contents

Chapter	Page
Chapter 1: Introduction	1
1.1. Overview	1
1.2. Game Tree Search; the Minimax Procedure	1
1.3. Alpha-beta Pruning and Depth-First Search	5
1.4. Breadth-First Search	7
1.5. Selective Search	8
1.6. Best-First Search and SSS*	9
1.7. B*: a Best-First Proof Procedure	10
1.8. Conspiracy Numbers	11
Chapter 2: Conspiracy Numbers	13
2.1. Overview	13
2.2. McAllester's Algorithm: Its Philosophy	13
2.3. How Conspiracy Numbers Works	15
2.4. The Algorithm	22
2.5. Minimal Sets	28
Chapter 3: Analysis of McAllester's Algorithm	30
3.1. Introduction	30
3.2. Preliminary Definitions and Observations	31
3.3. Minimality	34
3.4. The McAllester Tree	40

3.5. Optimality 62

Chapter 4: Improving McAllester's Algorithm 71

4.1. Introduction 71

4.2. The Improved Algorithm 71

4.3. The Success of CNEW 83

Chapter 5: Conclusions 85

5.1. Accomplishments of this Thesis 85

5.2. Suggestions for Further Work 86

References 88

List of Tables

Table	Page
Table 2.1: Conspiracy Numbers and conspirators	17
Table 3.1: Depth and number of nodes in Uniform-McAllester tree	61
Table 3.2: Depth and number of nodes in Uniform-McAllester tree	61
Table 4.1: Number of nodes in CNEW tree	82
Table 4.2: Number of nodes in CNEW tree	82

List of Figures

Figure	Page
Figure 1.1 Minimax tree	4
Figure 1.2 An alpha cutoff	5
Figure 1.3 A beta cutoff	6
Figure 2.1 Minimax tree	16
Figure 2.2 Conspiracy Numbers	18
Figure 2.3 Expansion of node E	20
Figure 2.4 Expansion of node D	21
Figure 3.1 Relationship diagram	34
Figure 3.2 Minimax trees	37
Figure 3.3 $\overline{P}_i$-tree	41
Figure 3.4 $\overline{P}_i$-trees	43
Figure 3.5 $\overline{P}_2$-tree for w = 3	44
Figure 3.6 An intermediate $\overline{P}_1$-tree	44
Figure 3.7 Lower bound of range changes	50
Figure 3.8 Range increases after expansion	51
Figure 3.9 $\overline{C}_i$-tree	52
Figure 3.10 Intermediate $\overline{C}_0$-tree	52
Figure 3.11 Intermediate $\overline{C}_{i-1}$-tree	53
Figure 3.12 $\overline{F}_{k+1}$-tree	63
Figure 3.13 Intermediate full tree	69
Figure 4.1 Tree	73

Chapter 1

Introduction

1.1. Overview

Computer game playing is an important area of artificial intelligence. Although mainstream research in artificial intelligence has moved away from this topic, game playing is still an interesting research problem.

This thesis discusses a new algorithm for searching game trees, the Conspiracy Numbers algorithm. The remainder of this chapter gives the reader the necessary background information. Section 1.2 introduces game tree search and the minimax procedure. Section 1.3 presents alpha-beta pruning and depth-first search. Later sections describe other methods of game tree search. The final section of this chapter briefly introduces the Conspiracy Numbers algorithm and highlights the accomplishments of this thesis.

1.2. Game Tree Search; the Minimax Procedure

This thesis is restricted to two person games. The players follow rules that are specific to the particular game; the rules determine the legal moves each player may make. Allowing only two players avoids the additional complexity caused by interaction and conflict of interest. As well, in the games considered in this thesis, it is assumed that each player knows completely what moves the other player has available to him at any stage of the game, as well as knowing his own possible moves. That is, attention is restricted to *perfect-information* games. Thus, games of chance are excluded because perfect-information does not exist. In most games, starting from the initial position and applying the rules repeatedly leads eventually to one of three possible outcomes: a win for the player who moves first, a win for the player who moves second, or a draw (although the discussion would also be valid for games that had different levels of winning: a win by 5 for one player would be better than a win by 1). There is an additional assumption in this thesis: a win for one player is a loss for the other player; similarly, a move that is advantageous for one person is as disadvantageous for the other. This type of game is known as a *zero-sum* game. This thesis is

restricted to two person, zero sum games with perfect information.

A game tree represents all possible moves from an initial position. The nodes of the tree are particular states or configurations in the game. The branches emanating from each node represent the legal moves; they represent all paths through the game tree. The interior nodes of the game tree represent positions with possible continuations, whereas the terminal nodes represent final positions in the game (those with no continuation). For terminal nodes, the value (win, loss, or draw) is known with certainty.

The goal in searching a game tree is to determine at a particular position a "good" move. The indicated move is then made; the opponent replies, and the search is repeated to find the next good move from the new position. When searching for a good move, because of time and storage constraints, it is usually possible to examine only a small part of the entire game tree; this part is called the *search tree*. The leaf nodes of the search tree may or may not be terminal nodes, depending on whether they correspond to terminal positions of the game.

To estimate the worth of a move, a static evaluation function is applied to the leaf nodes of the search tree. The value obtained by this function at a particular leaf node measures the worth (goodness) of the leaf node and is based on various features thought to influence its worth. For example, in chess, factors used in the evaluation function may include the number and type of pieces possessed by each player. As well, positional factors may be part of the evaluation function (control of the centre is usually important). It is customary in analyzing game trees to adopt the convention that the game positions favourable to one player cause the evaluation function to have a positive value (a win position for this player is usually given a value of $+\infty$), while positions favourable to the opponent cause the evaluation function to have a negative value (a win position for the opponent is usually given a value of $-\infty$). Values near zero correspond to positions neutral to both players (a draw is given the value zero).

It is assumed throughout this thesis that positions with positive evaluations are those favourable to the player whose turn it is to move first. It is this player's goal to choose the move with the maximum evaluation, i.e., the position most favourable to him; this player is called MAX. The objective of the other

player, the adversary, is to select the move with the minimum evaluation; this player is called MIN.

If the static evaluation function were exact, then one would need to evaluate only the immediate successors of the current position to find the right move. If this position occurs at a MAX node, choosing the successor with the maximum evaluation would give the best move. On the other hand, if this position occurs at a MIN node, choosing the successor with the minimum evaluation would give the best move. Indeed, a perfect evaluation function specifies whether a particular position leads to a win for either player.

Unfortunately, except for simple games, the static evaluation function provides only a rough estimate of the worth of a position. Therefore, further expansion is done, yielding a larger search tree. If it is assumed that the static evaluation function is more accurate when applied to deeper nodes than when applied to shallow nodes [1], a better procedure may be used to evaluate the successors of the current position than simply applying the evaluation function to the successors themselves. The method of backing up the static values of the leaves of a tree to obtain a value at the root of the tree is known as the *minimax procedure*. The minimax procedure works in the following way [Nils71]. If MAX were to choose among leaf nodes, he would choose the one with the highest static value; therefore, the maximum of the evaluations of the leaves is the value backed up to a node where MAX is to play. On the other hand, if MIN were to choose among leaf nodes, he would choose the one with the lowest static value; therefore, the minimum of the evaluations of the leaves is the value backed up to a node where MIN is to play. After values are assigned to the parents of all leaves, values are backed up, level by level in the same way, until successors of the root node have been assigned backed-up values. The root node is assumed to be a MAX node; thus the move chosen is the one corresponding to the successor of the root that has the largest backed-up value.

The minimax procedure received its name because MIN chooses a move having a value that is the minimum over all MAX's possible moves (i.e., the *minimum* of the *maximum*, since MAX's choice is the move with the maximum value).

---

1 This assumption will be discussed in Chapter 2.

In this thesis, a capital letter is used to denote the name of a node; its backed-up value is represented by the corresponding lower-case letter. A node whose value is denoted by b=4, for example, implicitly gives the node the name B.

An example of minimax search can be found in Figure 1.1 with MAX nodes shown with boxes and MIN nodes with circles. For this sub-tree of depth 2 and constant *width* or *branching factor* of 2 (each non-leaf has two children), a value of 4 is minimaxed back to the root of the sub-tree. This value is obtained after applying the static evaluation function to the leaves (nodes D, E, F and G). If applied directly to node A, the static evaluation function may have given a different value to node A.

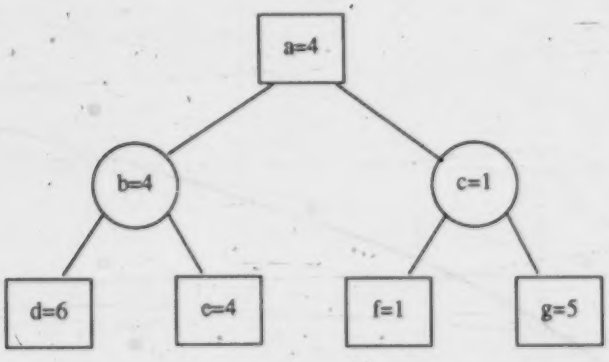


Figure 1.1. Minimax tree.

The minimax procedure provides a way of using information deep in the tree to obtain a more accurate value for an interior node than that obtained by applying the static evaluation function directly to an interior node. It does not, however, determine which nodes should be expanded when growing the search tree. There are various methods for growing a search tree. Shannon, in his seminal paper, visualized two types of game tree searches [Shan50]. One, *brute force* search, considered all possible moves to some fixed depth. The other *selective* search, used application-dependent knowledge to select nodes of the game tree to search. The later sections of this chapter will give the reader a more in-depth description of the two types of search. Sections 1.3 and 1.4 are devoted to two types of brute force search. The remaining sections

discuss various types of selective search.

1.3. Alpha-beta Pruning and Depth-First Search

Depth-first search [Nils71] is a brute force type of search. It expands the most recently generated node first until a predetermined depth or stopping condition (e.g., a terminal node) is reached. The aim of the search is to obtain a minimax value for the root of the search tree by studying all necessary nodes from the root to the predetermined depth. Alpha-beta pruning [KM75] is a method of avoiding fruitless computation during depth-first minimax search. In Figure 1.2, at node A, MAX always attains a value of at least 15. This lower bound for his achievement is called the **alpha** parameter. Since node D has a value 10, the value of node E need not be computed for the reason that if node E has a value greater than 10, then MIN will not choose this node. If its value is less than 10, MAX will choose node B instead of node C. Thus, node E can never influence the value at the root. Node E is said to be an **alpha cutoff.** This type of cutoff occurs at the MIN level when one child (here D) has a value less than alpha.

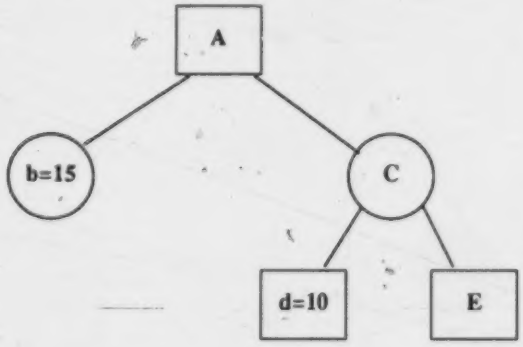


Figure 1.2. An alpha cutoff.

Similarly, in Figure 1.3, 20 is the upper bound for MAX at node B and G can never affect the result. Node G is said to be a **beta cutoff.** The **beta** parameter (the upper bound for MIN) is 20 and search at the MAX level quits when a child (here F) has a value greater than beta.

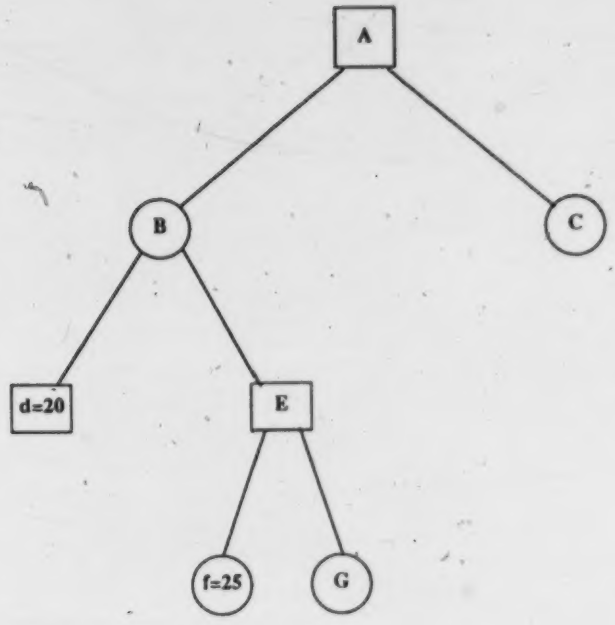


Figure 1.3. A beta cutoff.

The backward pruning used in adding alpha-beta cutoffs to the minimax procedure is able to offer a large saving in time without affecting the minimax value of node A. If a tree has a constant branching factor of w at non-leaf nodes and is of depth d, minimax search would examine w^d leaf nodes. Using alpha-beta pruning, Knuth and Moore [KM75] show that approximately $w^{d/2}$ nodes are examined in the best case[2]. The best case occurs when there is perfect ordering throughout the search tree; that is, the best node (for MAX, the node with the highest value; for MIN, the node with lowest value) is considered first at all interior nodes. Typically, heuristics are used to order the moves at interior nodes. The aim is to get the "best" move first and so build a smaller tree.

---

2 The exact formula is $w^{\left\lceil \frac{d}{2} \right\rceil} + w^{\left\lfloor \frac{d}{2} \right\rfloor} - 1$.

Depth-first search suffers from a major defect: it considers one path relentlessly until it reaches a terminal node or until a prescribed depth limit is reached. If the depth limit is chosen is small, then the backed-up minimax value is probably not as accurate as it would be if the depth limit were chosen deeper. However, if the depth limit is chosen too large, then resources, such as time, may expire before relevant nodes at a shallow depth have been considered. For this reason, ordinary depth-first search is often not a practical method for efficient search in applications in which time is important. Instead, *iterative deepening* [Mars87] is used in which a series of searches is carried out. It is a full-width search to increasing depths. Surprisingly, the cost of searching first to depth two, then repeating the search to depth three, etc., is feasible. A transposition table [Mars87] (a large hash table) is used to recognize positions seen before; this eliminates the need to search those positions again. Iterative deepening, when used with transposition tables, is viable because information from a particular depth allows an efficient sorting of the nodes so that alpha-beta cutoffs occur frequently when the depth is increased.

1.4. Breadth-First Search

Breadth-first search [Pear84] may be thought of as a search that works down the tree by expanding the nodes at a particular depth; at each depth it works from left to right. Thus, all branches are followed to about the same depth. Breadth-first search is guaranteed to find the shortest path to the solution, if a solution exists.

Like depth-first search, breadth-first search has significant problems associated with it. Since all unexpanded nodes must be saved, storage requirements are large. Another drawback occurs if the solution to the problem is quite long; to find any solution, breadth-first explores all paths of lesser length. If all solutions to a problem are at approximately the same depth, breadth-first is wasteful. In game tree search, there may not be a short path to a solution and the supposed advantage of finding the shortest path is only an illusion since studying the complete game tree is impossible in interesting games. A third major problem associated with breadth-first search is that it is impossible to use alpha-beta pruning with breadth-first search. It is for the latter reason that depth-first search (with iterative deepening) is a much more common brute force

search method than breadth-first search.

Thus, both depth-first and breadth-first searches have inherent flaws. As types of *brute force search*, they search all nodes (except those pruned by the alpha-beta technique) in a rigidly prescribed way until a solution is found or a preset depth limit or other terminating condition (such as time) is reached.

Another problem occurs often in brute force methods (as it does in all heuristic searches). Static evaluation functions usually have some degree of inaccuracy; therefore, it is possible that a position may not be as good or bad as its evaluation suggests. For example, at any particular cutoff in a search, one may give an overly-optimistic value to a specific position. This is because the search is not deep enough to see the inevitable loss of value lurking "just over the horizon". In chess, one may not realize that a line of play will result in the eventual loss of the queen; at the point where the search is discontinued, the line of play appears good. Self-delusion occurs simply because the search is not continued far enough to show the true outcome. Any search that quits before a solution is found is subject to the *horizon effect* [Berl73]. Brute force methods are particularly prone to this problem, since they use an artificial depth limit as a termination criterion. Perhaps, one should look at other methods to solve search problems.

1.5. Selective Search

When searching a game tree, instead of using a brute force approach, one might consider the pruning of nodes that, when application-dependent knowledge is used, appear to be inferior to other nodes. By only searching the "interesting" nodes, this *selective search* would allow searching some nodes to a greater depth than others. This is similar to the model human game players use.

At first glance, this idea appears sound. Yet, it has one major problem: it requires extensive application-dependent knowledge to make correct decisions. Even with this knowledge, selective search sometimes prunes nodes that should not be discarded (the knowledge is not perfect). If one searched further, the value of the pruned nodes might increase, or the value of other (non-pruned) nodes might decrease below the value of nodes that have been discarded. For example, in chess, losing a queen is usu-

ally bad; selective search might prune the move that results in this loss. Occasionally, however, giving up a queen by one player may lead eventually to a win for that player. Here, the move should not be pruned! Thus, selective search done in this way is not the ideal search method. It was used frequently in older chess programs, but as computers became more powerful, it was largely replaced by brute force search.

1.6. Best-First Search and SSS*

It has been shown that both depth-first and breadth-first searches expand nodes in a prescribed way: when a node is considered for expansion by the algorithm, it is either expanded or pruned; there are no half-way measures available. Brute force searches may waste time expanding nodes with little promise. Selective search, too, is not ideal, because it may prune nodes that turn out to be promising.

Best-first search is different from brute force search. Best-first search moves around the tree in a *data-dependent* way. The node chosen for expansion at any particular time is not selected by predetermined, data-independent rules. Instead, it is expanded if it is the *best* node available for expansion at that time. This lack of rigid structure gives best-first search an advantage over other methods. However, it also has an important disadvantage. It requires an inordinate amount of storage (like breadth-first search).

In 1979, Stockman [Stoc79] used best-first search in his algorithm SSS*[3]. It is a fixed-depth search with 2 phases. In phase 1, it generates the minimal alpha-beta tree by expanding every MAX node fully; at the MIN level, the best node is expanded fully, while the other MIN nodes have only one successor generated. In phase 2, all nodes that are not fully expanded are maintained in a sorted list (called the OPEN list). The node in the OPEN list with the highest value is removed from the list and expanded, possibly resulting in more nodes being added to the list. Minimaxing is done to back up a value to the root. Cutoffs arise as in alpha-beta pruning. It has been proved by Campbell that SSS* dominates alpha-beta in the sense that it does not evaluate any node ignored by alpha-beta and sometimes examines fewer nodes than alpha-beta [Camp81].

---

3 The name is derived from "state space search".

SSS*, in spite of its small improvement over alpha-beta, is not the successful algorithm Stockman hoped to create; the advantage of having slightly fewer nodes simply does not outweigh the disadvantage of the large increase in storage requirements (the OPEN list requires approximately $w^{d/2}$ entries). Furthermore, the algorithm runs slowly because the OPEN list must be maintained in sorted order. Thus, SSS* is not a practical solution to minimax search.

1.7. B*: a Best-First Proof Procedure

B* is another best-first search method. B* differs from all aforementioned search methods; instead of a single value, the static evaluation function gives a range of values. In all other search methods presented so far, the static evaluation function applied to non-terminal nodes displays some inaccuracy. The values obtained by the static evaluation function are all estimates, and errors in the static value get propagated up the tree. To overcome this problem, B* uses a range of values instead of a point value. B* is noteworthy because it is able to express uncertainty. As Berliner states in describing his B* algorithm [Berl79], "knowledge, rather than a set of arbitrary predefined limits can be used to terminate the search". B* assumes that a leaf's minimax value lies in the range of values given by the evaluation function. With this assumption, the algorithm runs until it *proves* that a move is best. This may not be practical, as mentioned in the following paragraph.

B* prunes nodes permanently; however, as a proof procedure, it never prunes a node that shouldn't be pruned as older selective search algorithms did. While B* solves the problem of the inaccuracy of the root's minimax value (the minimax value is **guaranteed** to lie in the backed-up range of values), the algorithm is plagued by problems in determining an upper and a lower bound for the range of values of each leaf node. If the bounds are chosen too wide, then the search grows and time becomes a problem. On the other hand, the bounds cannot be chosen too narrow, for the algorithm requires the range to contain the actual value of the terminal node. Thus, there is still an evaluation-function problem! The algorithm also assumes that all values within the range are equally likely. This assumption is inappropriate; some values should clearly be considered more probable than others. If the range of values were small, the problem

would be less significant, but to guarantee that the leaf's value lies within the range requires that it cannot be small! PB*, a modification of B* [Pala85], allows for various probability distributions, but these probabilities are introduced *a priori* and are not based on the particular tree.

1.8. Conspiracy Numbers

Instead of the extremes of brute force search and selective search, a compromise strategy may be used. This type of search selects interesting nodes for further expansion. The uninteresting nodes are not pruned (permanently ignored); instead, they are temporarily by-passed. Later development may imply that a node previously ignored should now be expanded. This type of search has been termed *selective deepening*. It allows for additional effort to be applied to "interesting" positions. It is found in algorithms such as the Singular Extension Algorithm [ACH88] as well as in the Conspiracy Numbers Algorithm that is the subject of this thesis.

McAllester's Conspiracy Numbers algorithm [McAl88], like B*, is noteworthy in that it also considers the probability that a node will attain a certain value. McAllester's algorithm is a more realistic search method than B* because it generates the probability of a node taking on a particular value *from the tree itself;* rather than using artificial probabilities like B* and PB*. B* requires a static evaluation function that returns a range of values that requires much application-dependent knowledge. McAllester's algorithm appears to be attractive, because it requires much less application-dependent knowledge than B*; it uses only the static evaluation function returning a single value.

Since the Conspiracy Numbers algorithm never prunes a node, it does suffer from the problem that storage requirements may be significant. However, different versions of the algorithm have been built that require less storage (but more time).

McAllester's algorithm has been chosen as the topic for this thesis because it is hoped that the algorithm, or modifications of it, could resolve some of the problems with other types of search. The reader has now seen the defects in the other methods and it should now be clear that better methods are desirable.

McAllester presented his ideas for improvement in his conspiracy numbers algorithm, but he did not analyze the algorithm thoroughly. McAllester's algorithm is a *new* algorithm; therefore, there is not much literature on the subject. Using an implementation by Klingbeil, Schaeffer [Scha89] provides experimental evidence that the algorithm has potential. When tested on 300 tactical chess positions, the algorithm performed as well as alpha-beta search programs. On the other hand, the program was less successful in positional chess, often failing to terminate and frequently exploring one path to an extreme depth. Schaeffer discusses enhancements to increase the speed and decrease storage requirements. Klingbeil and Schaeffer [Klin89] present different implementations of the algorithm to study time-space trade-offs. Several enhancements were suggested for the algorithm; when implemented, some were successful and some failed to improve the algorithm.

As hinted at already, the algorithm is different from previous search algorithms; therefore, it has many aspects that should be studied. This thesis investigates the complexity of the algorithm; it presents the size of the smallest tree expanded by McAllester's algorithm when the algorithm terminates. The number of nodes in this tree is found to grow exponentially with CT[4]; in fact, McAllester's algorithm is $O\left[\left(w^{\frac{1}{w-1}}\right)^{CT}\right]$. Focusing on performance in certain ideal situations provides some insight into the best case performance (lower bounds on costs) on more difficult and realistic problems.

This thesis also provides an enhancement to McAllester's algorithm. The smallest tree built by the improved algorithm grows quadratically with CT!

The remaining chapters of this thesis relate to McAllester's algorithm as follows. Chapter 2 describes McAllester's algorithm as well as giving an example to illustrate how the algorithm works. Chapter 3 analyzes McAllester's algorithm. An improvement to this algorithm is presented and analyzed in Chapter 4. Conclusions are found in chapter 5.

---

4 CT, the conspiracy threshold, is explained in Chapter 2.

Chapter 2

Conspiracy Numbers

2.1. Overview

In this chapter, McAllester's algorithm is described in detail. Section 2.2 presents further motivation for the underlying principles of the algorithm. Section 2.3 describes how McAllester's algorithm works. The algorithm itself is presented in Section 2.4. Finally, some key aspects of the algorithm that have a major effect on the nature of its performance are highlighted in Section 2.5.

2.2. McAllester's Algorithm: Its Philosophy

At any given stage of a game tree search, the object of a search algorithm is to find a good move (of course, the ultimate goal is to find a best move). As noted in Chapter 1, most algorithms use one of two methods to accomplish their goal: either they search to a predetermined fixed depth in the search tree (brute force) or they prune moves that appear bad and search only those that initially have promise (selective search). Recent search algorithms are based on the approach of selective deepening, whereby leaves of the search tree are selectively expanded and none are ever pruned. McAllester's algorithm falls within this newer class.

Like the other algorithms reviewed in Chapter 1, McAllester's algorithm uses a minimax procedure. This procedure has an important assumption: a more accurate evaluation of the root of the search tree is obtained by searching deeper (by expanding one or more leaves of the search tree). For minimax searches, however, it is sometimes true that **searching deeper consistently degrades the quality of a decision** [Pear84]. This is termed *pathology* by Nau [Nau83]. He says that although pathology has been observed in game trees, in practice it is not a problem; that is, in most interesting game trees, a deeper search gives a more accurate minimax value than a shallow search. Pearl says that pathology in game tree searching is hidden because searching deeper allows other more dominant factors to emerge because terminal nodes occur at many different levels of the tree [Pear84]. For example, features found 5 ply from the root could become

irrelevant if a win is found at 7 ply. Of course, if pathology is obscured by these other factors, it does not mean one should ignore the inaccuracy of the static evaluation function; the error in it can be propagated to the root of the tree.

In game tree search, it is reasonable to assume that searching deeper in a game tree will allow the selection of a move more accurately than is possible with a shallower search [Nau83]. One must, however, decide to stop the search at some point, since there are problems of time and/or space if the search continues too long. The method of determining when to stop expanding the search tree used in Conspiracy Numbers is different from the method used by other algorithms.

It is known that some situations in a game tree are less quiescent (or stable) than others. A quiescent position is one that "can be assessed accurately without further search" [Mars87]. Quiescence is important in searching. Continuing to search until the situation quiets down can cause significant alterations in the root's minimax value (call it t_{root}). It appears, therefore, that to determine t_{root} with reasonable confidence, one should attempt to search as deep as is necessary to achieve quiescence. This implies that some parts of the tree should be searched deeper than others. The search should terminate when t_{root} stabilizes. That is, searching should continue until t_{root} "would not change much if the tree were expanded further" [McAl85]. This is the philosophy behind McAllester's algorithm.

In McAllester's algorithm,

> "the stability of the root's minimax value is measured in terms of *conspiracy numbers* (CN). A conspiracy number measures the number of leaf nodes whose value must be changed to alter t_{root} by a certain amount. A change in the value of a certain set of leaf nodes is called a *conspiracy* between those leaf nodes. A conspiracy number measures the size of the conspiracy necessary to bring about a certain change in the root value; the more conspirators[5] needed for a given change the less likely the change" [McAl85].

If the number of conspirators necessary to bring about a certain change in the minimax value of the root is one, then a change in the value of only one leaf node can achieve this particular minimax value for the root. If, however, several conspirators are necessary to bring about a certain change in the root value, it implies

5 By *conspirators*, McAllester meant the node(s) involved in the conspiracy.

that the values of several leaf nodes must change to alter the minimax value of the root. Intuitively, McAllester's ideas make sense; it is more likely that further node expansion will cause a change to just one leaf node than to several leaf nodes.

McAllester's algorithm works using a *conspiracy threshold* (CT). It is considered unlikely that CT or more leaves will conspire to change another node's minimax value. That is, if the root has, at a particular stage of the search, a minimax value of t_{root}, any other minimax value for the root is considered likely if and only if it would require fewer than CT leaves to conspire to change t_{root} to that value. Thus, *a range of likely values* relative to CT may be obtained for the root. The algorithm runs until the range of likely values has been reduced to a single value; at this time, only one minimax value for the root is considered likely (relative to CT). The algorithm is then said to have converged (relative to CT). It is the CT parameter that allows one to obtain as stable a minimax value for the root as time and space will allow; a larger CT will expand more nodes and, therefore, give a more accurate minimax value.

2.3. How Conspiracy Numbers Works

Throughout this thesis, certain notation is adopted. A capital letter represents a node, e.g., T. A capital letter with the "bar" symbol represents a tree, e.g., $\bar{T}$. The root node of $\bar{T}$ is T. The minimax value pertains to the subtree rooted at a node (or it may be considered to pertain to that node); a lower case letter is used to represent the minimax value of the corresponding tree, e.g. t is the minimax value of $\bar{T}$ or of T. $\downarrow(T,v)$ is the number of conspirators required to decrease the minimax value of T to v. $\uparrow(T,v)$ is the number of conspirators required to increase the minimax value of T to v. $[t_{min},t_{max}]$ is the range of likely values of a tree $\bar{T}$ where t_{min} is the smallest minimax value of T such that $\downarrow(T,t_{min}) < CT$ and t_{max} is the largest minimax value of T such that $\uparrow(T,t_{max}) < CT$.

The following examples are similar to those given in [Scha89]. They are used to help explain McAllester's algorithm. A subtree rooted at node T is shown in Figure 2.1; it is assumed that any node with minimax value j is denoted by J. At the stage shown, node T has a minimax value of t = 4. This value

is obtained using the minimax procedure of Chapter 1. The conspiracy numbers algorithm must now decide whether to be satisfied with the minimax value of 4 for T or whether further expansion should be applied to obtain a minimax value for T that is more stable. To make this decision, the algorithm will check whether any other minimax value for T is likely; if so, the algorithm will continue expansion until only one minimax value for T is likely. The CN for T must be calculated to determine whether T's range of likely values is a single value.

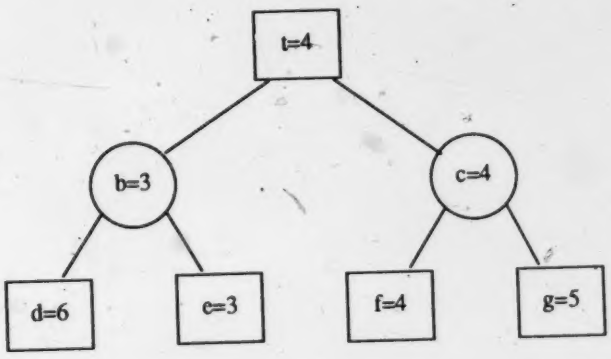


Figure 2.1. Minimax tree.

If $\bar{T}$ is a leaf node T, the CN for each value is either 1 or 0. If the node T has value t, no change must be done to give the node the value t; hence the conspiracy number for the value t is 0. For any other value, the conspiracy number is 1 because the leaf node itself (but no others) must change to give the node that particular value.

The conspiracy numbers of internal nodes are calculated by a simple method. To increase the minimax value of a MAX node, only one child needs to change its minimax value. The conspiracy number for that value v would thus be the *minimum* number of conspirators over all children (this would be the least effort). To decrease the MAX node to a value v, all children whose value is greater would have to be decreased. Thus, the number of conspirators is the *sum* of the number of conspirators of the children. At MIN nodes, similar reasoning leads to the reverse.

To illustrate the calculation of CN, look again at the minimax subtree shown in Figure 2.1 and study Table 2.1, which gives the conspiracy numbers for the node T. The table says that zero leaf nodes are required to change T's minimax value from 4 to 4 (since it already is the value 4); that two leaf nodes are required to change T's minimax value from 4 to 7 (namely, change to 7 the values of both D and E, or both

T's value	CN	Nodes to change
7	2	(D and E) or (F and G)
6	1	E
5	1	E or F
4	0	
3	1	F or G
2	2	(D or E) and (F or G)

Table 2.1. Conspiracy Numbers and conspirators.

F and G); and so on. A similar table can be constructed for each node of a tree.

To express mathematically the calculation of conspiracy numbers, break the conspiracy numbers column into two parts: the numbers required to increase T's minimax value a to a value v, $\uparrow$ (T,v), and those to decrease it to a value v, $\downarrow$ (T,v). Values v below T's minimax value t in $\uparrow$ (T,v) are assigned 0 as are values v above the minimax value in $\downarrow$ (T,v). For node T in Figure 2.1, $\uparrow$ (T,v) is $(2,1,1,0,0,0)^T$ for attaining $(7,6,5,4,3,2)^T$. For instance, for T to have the value 5, one node (E or F) must conspire. If e becomes 5, then b becomes 5 and t becomes 5. For t to become 7 two nodes must conspire ((D and E) or (F and G)). If d and e become 7, then b becomes 7 and t becomes 7. Similarly, $\downarrow$ (T,v) for node T is $(0,0,0,0,1,2)^T$ For brevity, these two vectors are combined (there is no ambiguity in doing so) in Table 2.1; the CN for T is written $(2,1,1,0,1,2)^T$. In Figure 2.2, associated with each node in the tree is an array of integers. The first column represents a list of potential minimax values of the node; the second column denotes the CN required to attain each of them.

The method of obtaining the CN of a leaf note T has already been discussed. Formally, it is written

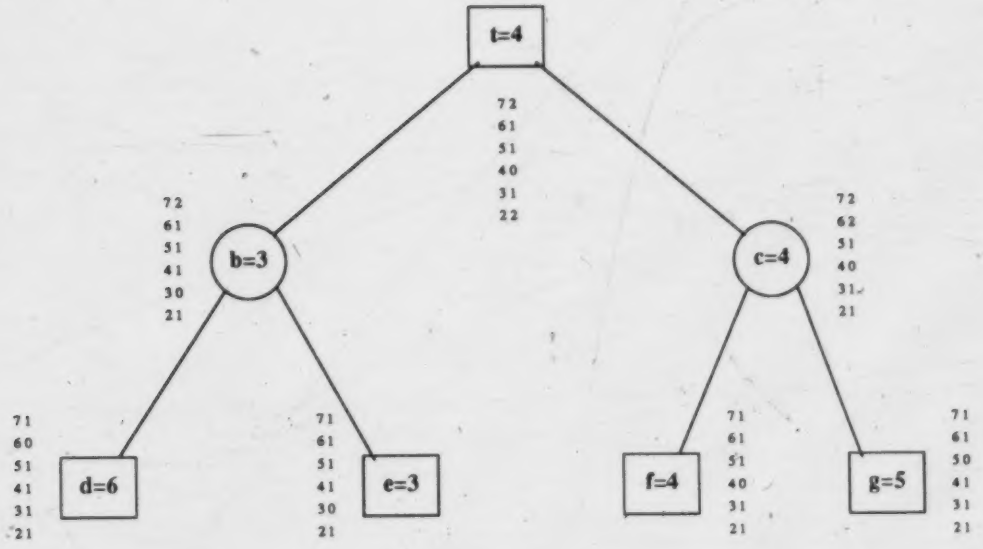


Figure 2.2. Conspiracy Numbers.

$\uparrow(T,t) = 0$

$\downarrow(T,t) = 0$

$\uparrow(T,v) = 1, v \neq t$

$\downarrow(T,v) = 1, v \neq t$

The following formulae may be used to obtain the CN of an interior node T from the CN of its children:

$\uparrow(T,t) = 0$

$\downarrow(T,t) = 0$

For a MAX node:

$$\downarrow(T,v) = \sum_{\text{all children } T_i} \downarrow(T_i,v), v < t,$$

$$\uparrow(T,v) = \min_{\text{all children } T_i} \uparrow(T_i,v),\ v > t$$

For a MIN node:

$$\downarrow(T,v) = \min_{\text{all children } T_i} \downarrow(T_i,v),\ v < t$$

$$\uparrow(T,v) = \sum_{\text{all children } T_i} \uparrow(T_i,v),\ v > t$$

To illustrate the use of these formulas, consider node B in Figure 2.2. Use the CN for $D(1,0,1,1,1,1)^T$ and the CN for $E(1,1,1,1,0,1)^T$. The CN for $B(2,1,1,1,0,1)^T$ is obtained as follows: Note that B is a MIN node with minimax value 3. Note that $\uparrow(D,v)$ is $(1,0,0,0,0,0)^T$, $\downarrow(D,v)$ is $(0,0,1,1,1,1)^T$, $\uparrow(E,v)$ is $(1,1,1,1,0,0)^T$, and $\downarrow(E,v)$ is $(0,0,0,0,0,1)^T$. Summing $\uparrow(D,v)$ and $\uparrow(E,v)$ yields $\uparrow(B,v)$ is $(2,1,1,1,0,0)^T$. Taking the minimum of $\downarrow(D,v)$ and $\downarrow(E,v)$ yields $\downarrow(B,v)$ is (0,0,0,0,0,1). Combining these two vectors produces the CN for B(2,1,1,1,0,1).

If the conspiracy threshold (CT) is 2, the range of likely values for T is [3,6] in the above example; this is seen in Table 2.1. This means it is considered unlikely that more than 2 leaf nodes will conspire; hence, it is improbable that the root will take on a value outside the range [3,6].

Given the conspiracy numbers, where is the search to continue to obtain useful information to help narrow the range of likely values of the root? The lower bound of the range of likely values is denoted t_{min}; the upper bound of the range is denoted t_{max}. McAllester's algorithm ignores values outside this range (they are considered unlikely to occur). Since in the example of Figure 2.2, there are more likely values above t_{root} than below (t_{root}=4 is further from t_{max}=6 than from t_{min}=3), the algorithm tries to change t_{root} to t_{max}. The goal is to narrow the range of likely values. One wishes to do as little work as possible since time is critical in searching. There appears to be less work to change node B to the value 6 than to change node C to 6 (the CN for 6 is 1 for B versus 2 for C). The algorithm descends to node B and then to node E (node D is already 6). Node E is a leaf node and gets expanded; the search tree becomes one ply deeper. E's children, nodes J and K, cause the minimax values and the CN to change for nodes E, B and T. The new values are shown in Figure 2.3. Now, the range of likely values of T becomes [4,6].

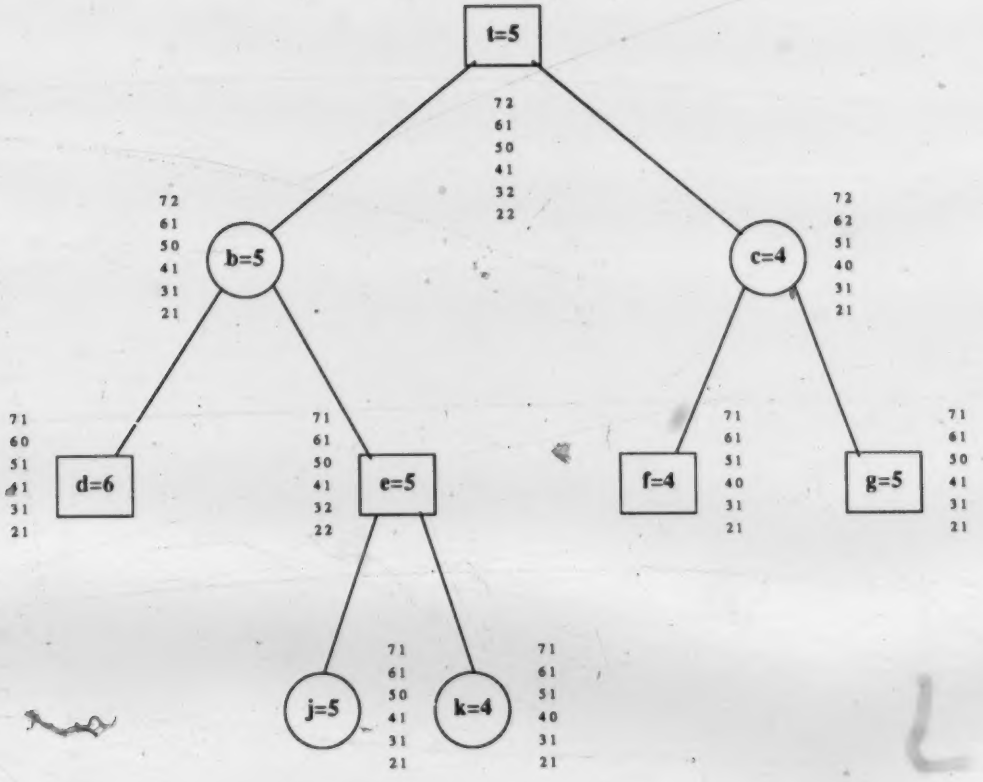


Figure 2.3. Expansion of node E.

Since 5 is equidistant from 4 and 6, arbitrarily, the algorithm tries to decrease the minimax value of the root to 4. Descending, node B is chosen (C already has the value 4). Descending further, D and E both have a CN of 1 for the value 4; the leftmost child, D, is arbitrarily chosen for expansion. The resulting tree is shown in Figure 2.4; note that the new nodes H and I alter the minimax value and CNs along the path D, B, T.

As mentioned already, at each expansion it is hoped to change t_{root} to t_{max} (or t_{min}). Even if unsuccessful, the range of likely values for t_{root} may still decrease in size. Unfortunately, this pleasant state of affairs does not always occur. Indeed, McAllester's algorithm does not always converge to a single likely

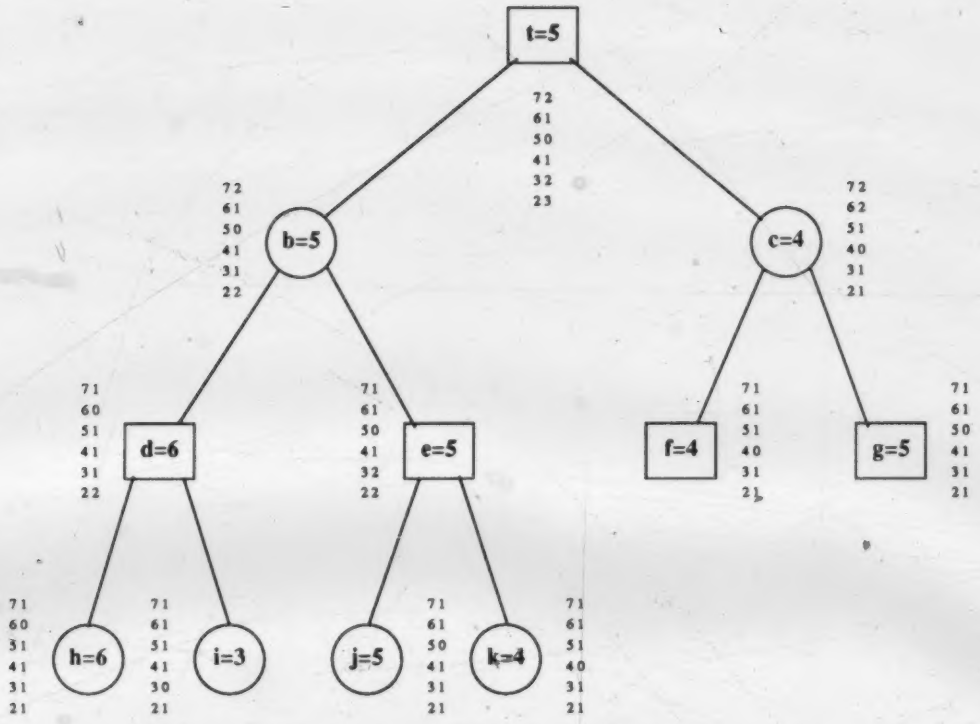


Figure 2.4. Expansion of node D.

value. This is a major problem with the algorithm. A *convergent* tree is the search tree produced when the range of likely values becomes a single value. McAllester's algorithm terminates successfully if a convergent tree is determined. Otherwise, the algorithm continues indefinitely or terminates unsuccessfully (due to time or other constraints) without constructing a convergent tree.

McAllester's algorithm is noteworthy in several respects. Instead of calculating only a point value for the root of a tree, it calculates a range of likely values. This range helps deal with the error in the static evaluation function. Each value has with it an associated *likelihood*. The root may become any value in the range of likely values since fewer than CT leaf nodes are required to conspire for this to occur. For the root to become any value outside the range is unlikely (but possible) because at least CT leaf nodes must

conspire. The CN for a value gives a good measure of the uncertainty; this piece of useful information is not usually available with other algorithms. An important advantage of the method is that the range of likely values is computed without additional application-dependent knowledge.

2.4. The Algorithm

McAllester's Conspiracy Numbers Algorithm is now presented.

McAllester's Algorithm

Procedure McAllester (CT)

M1: For a given tree $\bar{T}$,

(1) T is the root node,

(2) $T_1, \ldots, T_w$ are the children of T,

(3) $\bar{T}_1, \ldots, \bar{T}_w$ are the trees rooted at $T_1, \ldots, T_w$, respectively,

(4) t is the minimax value of T (or $\bar{T}$),

(5) $[t_{min}, t_{max}]$ is the range of likely values of T,

(6) v is some arbitrary value that the tree T may achieve for its minimax value. Note that in this description of the algorithm all values of v are permissible; however, in a practical implementation, v is selected for a reasonably small finite set.

M2: /* Initial tree consists of root node only */

$\bar{T} \leftarrow$ Root

$t_{min} \leftarrow -\infty$

$t_{max} \leftarrow +\infty$

M3: /* If $t_{min} = t_{max}$, convergence has occurred and algorithm will terminate. Note that this condition may never be satisfied; the algorithm may continue indefinitely. */

While ($t_{min} < t_{max}$) do

If $(t - t_{min}) < (t_{max} - t)$ /* select strategy */

Then $[\bar{T}, t, \uparrow(T,v), \downarrow(T,v)] \leftarrow$ IncreaseRoot($\bar{T}$, t_{max}, Maxnode) /*descend tree; expand one node */

Else $[\bar{T}, t, \uparrow(T,v), \downarrow(T,v)] \leftarrow$ DecreaseRoot($\bar{T}$, t_{min}, Maxnode) /*descend tree; expand one node */

End If

$t_{min} \leftarrow \min_{\text{all } v} \{v: \downarrow(T,v) < CT\}$ /* calculate lower bound */

$t_{max} \leftarrow \max_{\text{all } v} \{v: \uparrow(T,v) < CT\}$ /* calculate upper bound */

End While

End Procedure McAllester

Function DecreaseRoot($\bar{T}$, t_{min}, type)

D1: /* The subroutine DecreaseRoot attempts to eliminate t_{min} from the range of likely values of $\bar{T}$. DecreaseRoot expands one leaf node of $\bar{T}$. This expanded tree is returned as the new tree $\bar{T}$. Note that "type" refers to whether the root of $\bar{T}$ is a Maxnode or a Minnode. */

D2:

If $\bar{T}$=terminal node

Then

$\uparrow(T,t) = 0$

$\uparrow(T,v) = \infty$ if $v \neq t$

$\downarrow(T,t) = 0$

$\downarrow(T,v) = \infty$ if $v \neq t$

End If

If $\bar{T} \neq$terminal node and $\bar{T}$=leaf

Then For $1 \leq i \leq w$ do /* expand the w successors of $\bar{T}$ */

Determine t_i

$$\uparrow(T_i,v) \leftarrow \begin{cases} 1 \text{ if } v > t_i \\ 0 \text{ if } v \le t_i \end{cases}$$

$$\downarrow(T_i,v) \leftarrow \begin{cases} 1 \text{ if } v < t_i \\ 0 \text{ if } v \ge t_i \end{cases}$$

End For

End If

D3: /* Non-leaf which is Maxnode */

If $\bar{T} \neq$ leaf and type=Maxnode

Then

$M \leftarrow \{\bar{T}_i: \ t_i > t_{min}\}$ /* determine minimal set */

Select $\bar{T}_j$ to be the leftmost of all the children of $\bar{T}$ in M /* descend leftmost subtree */

$[\bar{T}_j, t_j, \uparrow(T_j,v), \downarrow(T_j,v)] \leftarrow$ DecreaseRoot($\bar{T}_j$, t_{min}, Minnode)

End If

D4: /* Non-leaf which is Minnode */

If $\bar{T} \neq$ leaf and type=Minnode

Then

$M \leftarrow \{\bar{T}_i: \ \downarrow(T_i, t_{min}) \le \downarrow(T_k, t_{min}), 1 \le k \le w\}$

Select $\bar{T}_j$ to be leftmost child of $\bar{T}$ in M /* determine minimal set; descend leftmost subtree */

$[\bar{T}_j, t_j, \uparrow(T_j,v), \downarrow(T_j,v)] \leftarrow$ DecreaseRoot($\bar{T}_j$, t_{min}, Maxnode)

End If

D5: /* Recompute $\uparrow(T,v)$ and $\downarrow(T,v)$ */

If type = Maxnode

Then

$$\uparrow(T,v) \leftarrow \begin{cases} \min\limits_{1 \le i \le w} \{\uparrow(T_i,v)\} & \text{if } v>t \\ 0 & \text{if } v \le t \end{cases}$$

$$\downarrow(T,v) \leftarrow \begin{cases} \sum\limits_{1 \le i \le w} \downarrow(T_i,v) & \text{if } v<t \\ 0 & \text{if } v \ge t \end{cases}$$

Else

$$\downarrow(T,v) \leftarrow \begin{cases} \min\limits_{1 \le i \le w} \{\downarrow(T_i,v)\} & \text{if } v<t \\ 0 & \text{if } v \ge t \end{cases}$$

$$\uparrow(T,v) \leftarrow \begin{cases} \sum\limits_{1 \le i \le w} \uparrow(T_i,v) & \text{if } v>t \\ 0 & \text{if } v \le t \end{cases}$$

End If

D6: /* Recompute t */

If type=Maxnode

Then $t \leftarrow \max\limits_{1 \le i \le w} \{t_i\}$

Else $t \leftarrow \min\limits_{1 \le i \le w} \{t_i\}$

End If

D7: /* Return */

Return $[\bar{T}, t, \uparrow(T,v), \downarrow(T,v)]$

End DecreaseRoot

Function IncreaseRoot($\bar{T}$, v_{max}, type)

I1: /* The subroutine IncreaseRoot attempts to eliminate t_{max} from the range of likely values of $\bar{T}$. IncreaseRoot is the dual of DecreaseRoot. */

I2:

If $\bar{T}$=terminal node

Then

$\uparrow(T,t) = 0$

$\uparrow(T,v) = \infty$ if $v \neq t$

$\downarrow(T,t) = 0$

$\downarrow(T,v) = \infty$ if $v \neq t$

End If

If $\bar{T}$≠terminal node and $\bar{T}$=leaf

If $\bar{T}$=leaf

Then For $1 \leq i \leq w$ do /* expand the w successors of $\bar{T}$ */

Determine t_i

$$\uparrow(T_i,v) \leftarrow \begin{cases} 1 \text{ if } v > t_i \\ 0 \text{ if } v \leq t_i \end{cases}$$

$$\downarrow(T_i,v) \leftarrow \begin{cases} 1 \text{ if } v < t_i \\ 0 \text{ if } v \geq t_i \end{cases}$$

End For

End If

I3: /* Non-leaf which is Minnode */

If $\bar{T} \neq$ leaf and type=Minnode

Then

$M \leftarrow \{\bar{T}_i: t_i < t_{max}\}$ /* determine minimal set */

Select $\bar{T}_j$ to be the leftmost of all the children of $\bar{T}$ in M /* descend leftmost subtree */

$[\bar{T}_j, t_j, \uparrow(T_j,v), \downarrow(T_j,v)] \leftarrow$ IncreaseRoot($\bar{T}_j$, t_{max}, Maxnode)

End If

I4: /* Non-leaf which is Maxnode */

If $\bar{T} \neq$ leaf and type=Maxnode

Then

$M \leftarrow \{\bar{T}_i: \uparrow(T_i, t_{max}) \leq \uparrow(T_k, t_{max}), 1 \leq k \leq w\}$

Select $\bar{T}_j$ to be leftmost child of $\bar{T}$ in M /* determine minimal set; descend leftmost subtree */

$[\bar{T}_j, t_j, \uparrow(T_j,v), \downarrow(T_j,v)] \leftarrow$ IncreaseRoot($\bar{T}_j$, t_{max}, Minnode)

End It

I5: /* Recompute $\uparrow(T,v)$ and $\downarrow(T,v)$ */

If type = Maxnode

Then

$$\uparrow(T,v) \leftarrow \begin{cases} \min\limits_{1 \leq i \leq w} \{\uparrow(T_i,v)\} \text{ if } v > t \\ 0 \text{ if } v \leq t \end{cases}$$

$$\downarrow(T,v) \leftarrow \begin{cases} \sum\limits_{1 \leq i \leq w} \downarrow(T_i,v) \text{ if } v < t \\ 0 \text{ if } v \geq t \end{cases}$$

Else

$$\downarrow(T,v) \leftarrow \begin{cases} \min\limits_{1 \leq i \leq w} \{\downarrow(T_i,v)\} \text{ if } v < t \\ 0 \text{ if } v \geq t \end{cases}$$

$$\uparrow(T,v) \leftarrow \begin{cases} \sum\limits_{1 \leq i \leq w} \uparrow(T_i,v) \text{ if } v > t \\ 0 \text{ if } v \leq t \end{cases}$$

End If

I6: /* Recompute t */

If type=Maxnode

Then $t \leftarrow \max_{1 \le i \le w} \{t_i\}$

Else $t \leftarrow \min_{1 \le i \le w} \{t_i\}$

End If

I7: /* Return */

Return $[\bar{T}, t, \uparrow(T,v), \downarrow(T,v)]$

End IncreaseRoot

2.5. Minimal Sets

Certain aspects of the algorithm play an important role in the analysis presented in Chapters 3 and 4 of this thesis. The current discussion is presented to prepare the reader for the subsequent development. The discussion is restricted to observations about the subroutine DecreaseRoot; dual observations hold for IncreaseRoot.

In Step D3, if the root T is an interior Maxnode, the *minimal set* is selected; the algorithm will descend the left-most subtree. This set is the children of T that have minimax values greater than t_{min}. The minimax value of *all* nodes in the minimal set must decrease to values less than or equal to t_{min} for the minimax value of T to become less than or equal to T_{min}. In McAllester's algorithm, the left-most node in this set is always selected for expansion. In Chapter 3, it is shown that choosing the node for expansion in this way leads to a tree with an extremely deep left branch. In Chapter 4, a modified algorithm, CNEW, is presented that is more discretionary in selecting a node for expansion from the minimal set. As a consequence, the tendency is no longer to extend the tree down the leftmost branch but rather to build a broader

tree. For a particular static evaluation function (one in which all the leaves have the same value), the modified algorithm is shown to build a tree with fewer nodes than McAllester's algorithm. It is conjectured that this is true in general for reasons that are stated in Chapter 4.

In McAllester's algorithm, after selecting the node in Step D3, DecreaseRoot strategy is called again with this selected node as the root of the tree $\bar{T}$. In this next invocation of DecreaseRoot, in Step D4, if T is an interior Minnode, the *minimal set* is selected in a different way. Here, it is the set of children of T such that no other child requires fewer conspirators to change its minimax value to t_{min}. Changing the minimax value of any node in the minimal set to a value less than t_{min} will cause a corresponding decrease in the minimax value of T. The leftmost node in this set is selected and DecreaseRoot strategy is called again with this new node as the root of the tree $\bar{T}$. Selection of a node for expansion in Step D4 in this way does not appear to have the same adverse effect on the rapid growth of the tree as does the selection in Step D3. Therefore, in the new algorithm, CNEW, the selection criterion in Step D_4 is left unaltered.

Note that in Step D4, the minimal set consists of a collection of tied contenders (each $\bar{T}_i$ in this set requires the same number of conspirators to change its minimax value to t_{min} and there is no $\bar{T}_i$ that can accomplish this with fewer conspirators) and is usually just one node. The selection criterion is then simply a mechanism of breaking this tie. Modifying the tie breaking mechanism is unlikely to have a major effect on the algorithm's performance. On the other hand, in Step D3, the minimal set consists of nodes whose conspiracy numbers to achieve a particular value can differ dramatically. The selection criterion in McAllester's algorithm (selecting the leftmost node) appears to be too simplistic. The algorithm CNEW attempts to select that node for the minimal set which achieves the desired change (alter t to t_{min}) with the minimum number of node expansions.

Chapter 3

Analysis of McAllester's Algorithm

3.1. Introduction

This chapter analyzes McAllester's algorithm as described in Chapter 2. In an effort to estimate the complexity of the tree produced by McAllester's algorithm, lower bounds for the number of nodes in such a tree are obtained. The number of leaf nodes in a search tree is a commonly used measure of the cost of a search algorithm for selecting a move [SD69, KM75, Baud78]. Of course, the number of nodes is not directly proportional to the cost (processing time and storage requirements) of the algorithm since other factors are also relevant. For example, some nodes may be more costly to expand than others, or node expansion may become costlier as the number of nodes increases. Nevertheless, the cost of a search algorithm is a function of its computing environment and of the number of nodes in the search tree that it constructs. Thus, an analysis of the number of nodes of a search tree obtained by an algorithm is a first step in theoretically estimating its cost.

A lower bound analysis rather than a complete analysis of the number of nodes of McAllester's algorithm is undertaken for several reasons. First, because of the pathology of static evaluation functions in general [Nau83], other means of analysis, such as average cost, are difficult tasks [Li89]; indeed, for an arbitrary static evaluation function, McAllester's algorithm may not converge at all. Second, lower bounds, when they are tight, can provide insight into the behavior of the algorithm [KM75] and are often the first step in a complete cost analysis. Third, lower bound complexity analysis allows comparison of McAllester's algorithm with other tree-searching algorithms for which lower bound analyses are also available. If the analysis shows that McAllester's algorithm is significantly better in this respect than other algorithms, its development might be vigorously pursued; on the other hand, if it is only competitive, perhaps McAllester's algorithm can be modified or combined as a hybrid with the best features of other algorithms to yield a better algorithm. If the lower bounds for McAllester's algorithm are comparatively large, it is possible that the algorithm would be dropped as a viable practical alternative.

In this thesis, it is shown that a tight lower bound for McAllester's algorithm is obtained when the static evaluation function is constant (that is, the static value at all leaf nodes is the same) Unlike the case in which the static evaluation function is arbitrary, it is proven that McAllester's algorithm *always converges when the static evaluation function is constant.*

Throughout this chapter, four assumptions are made.

1) If one child of a node is present, then all children are evaluated. This assumption is made to be consistent with McAllester's algorithm.

2) Every leaf node can be expanded; that is, expansions can be made indefinitely. Including terminal nodes does not change the analysis.

3) It is assumed that CT > 1; otherwise, McAllester's algorithm converges trivially with a single node.

4) Any tree is rooted at a MAX node, unless otherwise specified.

3.2. Preliminary Definitions and Observations

For a given CT > 1 and an arbitrary static evaluation function, some immediate questions arise. Does a convergent tree exist and, if so, does McAllester's algorithm construct it? These are interesting and difficult questions, but not ones that for arbitrary static evaluation functions are pursued in this thesis. Instead, attention is focused on only those problems for which McAllester's algorithm is successful in constructing a convergent tree. The reason for this apparent restriction will be discussed below, following Definition 3.2.

Definition 3.1: Given a conspiracy threshold CT and some static evaluation function, a **McAllester tree** is the tree produced by McAllester's algorithm *when convergence occurs.*

□

It is desirable to determine, for a given CT and a given static evaluation function, the number of nodes in the McAllester tree. This objective is difficult to achieve (not possible unless constraints are placed on the static evaluation function) and instead meaningful lower bounds are obtained. For a given CT and given

static evaluation function, consider the set of all trees that are convergent. Note that if a McAllester tree exists, then this set is not empty (a McAllester tree is a convergent tree).

Definition 3.2: Given a conspiracy threshold CT and some static evaluation function, the **minimal tree** is a convergent tree with the least number of nodes.

□

Analysis of the minimal tree is meaningful in two ways. First, the number of nodes in the minimal tree provides a lower bound on the number of nodes in a McAllester tree. Second, the minimal tree is the optimal convergent tree (with respect to the number of nodes) and provides a benchmark for the comparison of the effectiveness of algorithms (including McAllester's) which construct a convergent tree.

Now reasons are given for restricting study to convergent trees. This was done only to ensure the existence of a minimal tree. In other respects, the restriction does not affect later discussions and results in this chapter. The lower bound analysis is as valid when McAllester's algorithm does not produce a convergent tree as it is when it does!

For an arbitrary static evaluation function, this thesis does not succeed in determining the minimal tree. Constraints must be placed on the static evaluation function before significant results can be obtained (an interesting research problem). Nevertheless, the notion of a minimal tree is important to the presentation; it is used as a stepping stone for obtaining lower bounds for McAllester's algorithm. That is, lower bounds for the minimal tree provide lower bounds (although not as sharp) for the McAllester tree.

For a given CT, lower bounds on the number of nodes in a minimal tree are obtained by comparing it with a convergent tree for the same CT but for which the static evaluation function is held constant; that is, all leaf nodes have the same value. Restricting the static evaluation function in this way places the maximum constraint on it. With this constraint, for any given CT, however, it now becomes possible to show the existence of convergent trees (cf., Corollary 3.20).

Definition 3.3. Given a CT and a constant static evaluation function, a **uniform-minimal tree** is a convergent tree with the minimum number of nodes.

□

In Section 3.3, it is shown that any convergent tree (including the minimal tree) remains convergent if all static values of the leaf nodes are replaced by some single uniform value. Included in the set of convergent trees for a given CT and uniform static value is the tree constructed by McAllester's algorithm.

Definition 3.4: Given a CT, the **uniform-McAllester tree** is the tree constructed by McAllester's algorithm when the static evaluation function is a constant.

□

It is shown in Section 3.5 that the uniform-McAllester tree is far from optimal. Whereas the number of nodes in uniform-McAllester trees grows exponentially with CT (cf., Corollary 3.27), for a constant static evaluation function convergent trees are given which grow quadratically with CT. In Chapter 4, a modification is provided to McAllester's algorithm that experimentally achieves this modest quadratic growth. The discussion in Section 3.5 motivates the conjecture (which is not proven) that for a constant static evaluation function, the modified algorithm of Chapter 4 is optimal.

A consequence of this difference in growth rates of the uniform-McAllester tree and the uniform-minimal tree, for given CT, is that lower bounds on the uniform-minimal tree are unlikely to be useful lower bounds for the McAllester tree (they are far from tight). In Section 3.4, however, a somewhat surprising result is shown: a McAllester tree contains a uniform-McAllester tree. Thus, for given CT, the number of nodes in a uniform-McAllester tree (which grows exponentially with CT) provides a tight lower bound on the number of nodes in any McAllester tree (for the same CT and any static evaluation function).

Summarizing, the results of the remaining sections are concerned with establishing, for a given CT, the relationships depicted in Figure 3.1. The result that the number of nodes in a McAllester tree is

bounded below by the number of nodes in a minimal tree is an immediate consequence of Definition 3.1 and Definition 3.2. The result that the number of nodes in a uniform-minimal tree provides lower bounds on the number of nodes in a minimal tree is the subject of Section 3.3. In Section 3.4, the number of nodes in the uniform-McAllester tree is determined and it is shown that it provides lower bounds on the McAllester tree. The remaining relationship, namely that the number of nodes in the uniform-minimal tree is a lower bound on the uniform-McAllester tree is a direct consequence of Definitions 3.3 and 3.4. In Section 3.5, the tightness (more accurately, the looseness) of the lower bound is explored.

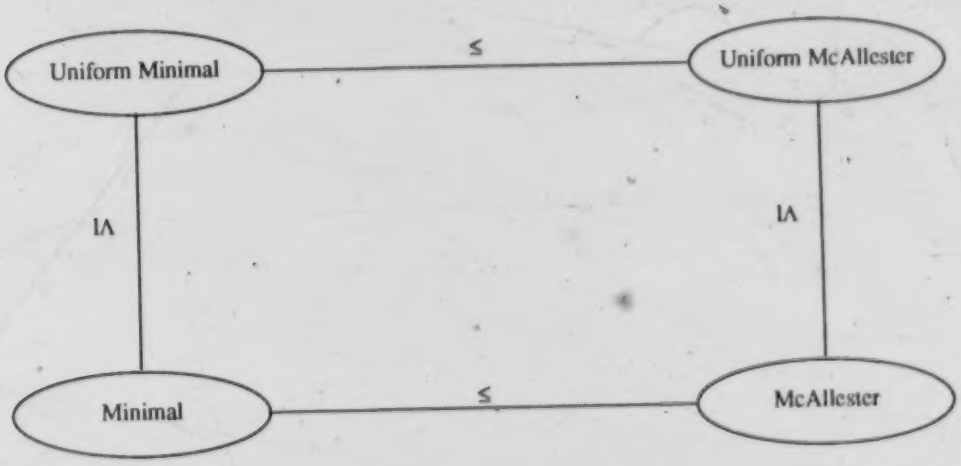


Figure 3.1. Relationship diagram.

3.3. Minimality

For a given CT, consider all convergent trees for which the static evaluation function is a constant and also all those convergent trees for which the static evaluation function is arbitrary but fixed. In this section, it is shown that the tree with the minimum number of nodes in the former case (the uniform-minimal tree) cannot contain more nodes than any tree (including the minimal tree) in the latter case. That is, it is shown that the number of nodes in the uniform-minimal tree is a lower bound on the number of nodes in

the minimal tree.

The first result necessary is found in the following lemma.

Lemma 3.5. Let the minimax value of a tree $\bar{T}$ be t. Then changing the static value s of a leaf node to s' changes the minimax value of $\bar{T}$ to a value in $[s', t]$ (or $[t, s']$).

Proof: Without loss of generality, only $[s', t]$ is considered. Consider the unique path from the root of $\bar{T}$ to the leaf node whose static value is s. Denote the nodes along this path by T_k, $k = 0, \ldots, d$, where T_k is at depth k. Let the corresponding minimax values at these nodes be t_k, $k = 0, \ldots, d$. Thus, $t_0 = t$ and $t_d = s$.

Clearly, changing s to s' in this tree $\bar{T}$ affects only the minimax values t_k at the nodes T_k, $k = 0, \ldots, d$. Denote the new minimax value at T_k by t_k', $k = 0, \ldots, d$. In particular, note that $t_d' = s'$.

It is asserted that for $k = d, d-1, \ldots, 0$, t_k' lies between s' and t_k. Thus, for $k = 0$, the assertion states that t_0' lies between s' and t_0 which is precisely the assertion of the lemma. The proof is by induction. The assertion is clearly true when $k = d$. Next, assume that the assertion is true for $k = d, d-1, \ldots, i$. In particular, assume that at the node T_i, the new minimax value t_i' lies between s' and t_i. It is shown that the assertion is true at T_{i-1}. Note that T_i is one of the w children of T_{i-1}. Denote the minimax value of the other w-1 children of T_{i-1} by $r_1, \ldots, r_{w-1}$; these are not affected by the change in the static value of T_d.

If T_{i-1} is a MAX node, then

$$t_{i-1} = \max \{t_i, \max_{1 \le j \le w-1} \{r_j\}\} \tag{3.1}$$

and

$$t_{i-1}' = \max \{t_i', \max_{1 \le j \le w-1} \{r_j\}\}. \tag{3.2}$$

By the inductive hypothesis, t_i' lies between s' and t_i; it then follows from (3.1) and (3.2) that t_{i-1}' lies between s' and t_{i-1}. Specifically, let

$$\mu = \max_{1 \le j \le w-1} \{r_j\}. \tag{3.3}$$

From the inductive hypothesis, at least one of $s' \le t_i' \le t_i$, or $t_i \le t_i' \le s'$, must be true. If $s' \le t_i' \le t_i$, then

$$s' \le \max\{s', \mu\} \le \max\{t_i', \mu\} \tag{3.4}$$
$$= t_{i-1}'$$
$$= \max\{t_i', \mu\} \le \max\{t_i, \mu\} = t_{i-1}.$$

If $t_i \le t_i' \le s'$, then there are two cases to consider; either $s' \ge \mu$, or $s' < \mu$. In the first case, where $s' \ge \mu$,

$$t_{i-1} = \max\{t_i, \mu\} \le \max\{t_i', \mu\} \tag{3.5}$$
$$= t_{i-1}'$$
$$= \max\{t_i', \mu\} \le \max\{s', \mu\} = s'.$$

In the second case, where $s' < \mu$,

$$t_{i-1}' = \max\{t_i', \mu\} \tag{3.6}$$
$$= \mu$$
$$= \max\{t_i, \mu\}$$
$$= t_{i-1}.$$

Thus, when T_{i-1} is a MAX node, the assertion that t_{i-1}' lies between s' and t_{i-1} has been verified.

Similarly, when T_{i-1} is a MIN node, then

$$t_{i-1} = \min\{t_i, \min_{1 \le j \le w-1}\{r_j\}\} \tag{3.7}$$

and

$$t_{i-1}' = \min\{t_i', \min_{1 \le j \le w-1}\{r_j\}\}. \tag{3.8}$$

Again, using (3.7), (3.8) and the inductive hypothesis, it follows that t_{i-1}' must lie between s' and t_{i-1}.

□

Example 3.6: Lemma 3.5 raises the question whether changing the static value s of a leaf node to s' will ever change the minimax value of $\bar{T}$ to a value other than s' or t. The answer to this question is affirmative; an illustration is provided in Figure 3.2. In this figure, when the static value s = 4 is replaced by the static value $s' = 1$, the minimax value changes from t = 4 to 2 (a value between $s' = 1$ and t = 4).

□

Corollary 3.7. Let the minimax value of a tree $\bar{T}$ be t. Changing the static value of a leaf node from its present value to t does not alter the minimax value of $\bar{T}$.

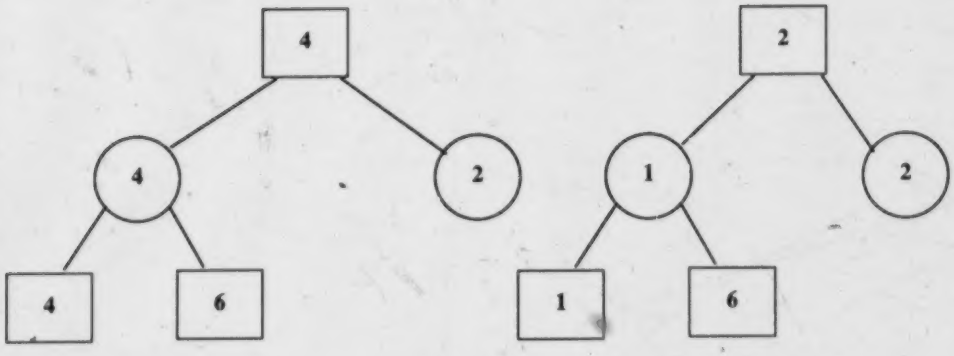


Figure 3.2. Minimax trees.

Proof: Substitute t for s in Lemma 3.5.

□

Theorem 3.8. A convergent tree remains convergent when the values of its leaves are replaced by some uniform static value.

Proof: Let $\bar{T}$ be a convergent tree whose minimax value is t and whose root is T. This theorem is proved by changing the static value of one leaf at a time to t. After changing the static value of one leaf to t, it is shown that the new tree, $\bar{T}'$, remains convergent. Since $\bar{T}$ has a finite number of leaves, it follows that the tree $\bar{T}$ remains convergent when that static value of all its leaves are changed to t. Since convergence is shift-invariant (i.e., convergence is not affected by adding a constant to the static evaluation function), convergence is still present where the static values of all leaves assume any uniform value.

As before, changing the static value to t of a leaf node of T affects the minimax values only along the (unique) path from the altered leaf to the root. Denote the nodes along this path by T_k, k = 0, ... , d, where T_k is at depth k and let the minimax value of T_k be t_k. Also, for the new tree $\bar{T}'$, denote the node corresponding to T_k by T_k' and let its new minimax value be t_k'.

It is asserted that, for $k = d, d - 1, \ldots, 0$,

$$\uparrow(T_k, t+1) \le \uparrow(T_k', t+1) \tag{3.9}$$

and

$$\downarrow(T_k, t-1) \le \downarrow(T_k', t-1). \tag{3.10}$$

As a consequence, it then follows that the minimum number of conspirators required to change the minimax value of T_0' to $t + 1$ cannot be less than the minimum number of conspirators required to change the minimax value of T_0 to $t + 1$. As well, the minimum number of conspirators required to change the minimax value of T_0' to $t - 1$ cannot be less than the minimum number of conspirators required to change the minimax value of T_0 to $t - 1$. Since Corollary 3.7 states that the minimax value of $\bar{T}'$ remains at t, it then also follows that the new tree remains convergent.

The proof of (3.9) and (3.10) is by induction. For the initial case $k = d$, first observe that because T_d is a leaf node,

$$\uparrow(T_d, t+1) \le 1 \tag{3.11}$$

and

$$\downarrow(T_d, t-1) \le 1. \tag{3.12}$$

Secondly, the minimax value of T_d' is $t_d' = t$ and, therefore,

$$\uparrow(T_d', t+1) = 1 \tag{3.13}$$

and

$$\downarrow(T_d', t-1) = 1. \tag{3.14}$$

From (3.11), (3.12), (3.13), and (3.14), it follows that the assertions (3.9) and (3.10) are true for $k = d$.

Working up the given path of the tree, inductively assume that the assertion is true for $k = d, d-1, \ldots, i$. That is, assume

$$\uparrow(T_i, t+1) \le \uparrow(T_i', t+1) \tag{3.15}$$

and

$$\downarrow(T_i, t-1) \le \downarrow(T_i', t-1). \tag{3.16}$$

It is required to show that

$$\uparrow(T_{i-1}, t+1) \le \uparrow(T'_{i-1}, t+1) \tag{3.17}$$

and

$$\downarrow(T_{i-1}, t-1) \le \downarrow(T'_{i-1}, t-1). \tag{3.18}$$

Note that T_{i-1} has w children, one of which is T_i. Denote the other w-1 children by R_j, j = 1, ... , w-1. These nodes R_j are unaffected by the change in the static value of T_d. If T_{i-1} (which is also T'_{i-1}) is a MAX node, from (3.15), it follows that

$$\begin{aligned} \uparrow(T_{i-1}, t+1) &= \min\{\uparrow(T_i, t+1), \min_{1\le j\le w-1}\{\uparrow(R_j, t+1)\}\} \\ &\le \min\{\uparrow(T'_i, t+1), \min_{1\le j\le w-1}\{\uparrow(R_j, t+1)\}\} \\ &= \uparrow(T'_{i-1}, t+1) \end{aligned} \tag{3.19}$$

and, from (3.16), that

$$\begin{aligned} \downarrow(T_{i-1}, t-1) &= \downarrow(T_i, t-1) + \sum_{1\le j\le w-1} \downarrow(R_j, t-1) \\ &\le \downarrow(T'_i, t-1) + \sum_{1\le j\le w-1} \downarrow(R_j, t-1) \\ &= \downarrow(T'_{i-1}, t-1). \end{aligned} \tag{3.20}$$

This proves the assertion for the case when $T_{i-1} = T'_{i-1}$ is a MAX node. If $T_{i-1} = T'_{i-1}$ is a MIN node, dual arguments yield a similar conclusion.

□

The significance of Theorem 3.8 relative to the complexity diagram depicted in Figure 3.1 is expressed by the following corollary.

Corollary 3.9. The number of nodes in a minimal tree for a given CT and some static evaluation function is bounded below by the number of nodes in a uniform-minimal tree (same CT and constant static evaluation function).

Proof: From Theorem 3.8, the minimal tree remains convergent when its static values are replaced by a constant. The number of nodes in the resulting tree cannot be fewer than the number in the minimal

tree (cf., Definition 3.3).

□

3.4. The McAllester Tree

In this section, attention is shifted away from the question of minimality in order to examine the behavior of McAllester's algorithm. The primary goal is to obtain a good lower bound for the number of nodes in the McAllester tree. Of course, the number of nodes in the uniform-minimal tree provides such a lower bound; but Section 3.5 shows that it does not yield as tight a bound as the one obtained in this section. It is shown that McAllester's algorithm builds a McAllester tree in three stages.

Stage I: McAllester's algorithm first builds a tree for which the range of likely values is reduced from $[-\infty, +\infty]$ to $[e,\infty]$ for some finite e. Such a tree is denoted by $\bar{P}_m$ below.

Stage II: McAllester's algorithm next expands the $\bar{P}_m$-tree to yield a tree for which the range of likely values is reduced from $[e, +\infty]$ to [b, d] for some finite b and d. Such a tree is denoted by $\bar{C}_m$ below. For a constant static evaluation function, it will be shown that the $\bar{C}_m$-tree is convergent; here, the algorithm terminates upon completion of Stage II.

Stage III: For an arbitrary static evaluation function, McAllester's algorithm continues expanding the $\bar{C}_m$-tree until convergence occurs (of course, the algorithm may never terminate and a McAllester tree may never be produced; recall that a McAllester tree was defined as a *convergent* tree in Definition 3.1).

First, the tree produced during Stage I of McAllester's algorithm is studied. A $\bar{P}_i$-tree, i = 0, ... , m, is defined and discussed and it is shown that McAllester's algorithm always builds these trees for some integer m yet to be determined.

Definition 3.10: A **$\bar{P}_i$-tree** is defined recursively by Figure 3.3, where P_i is the root node and B_2,..., B_w are leaf nodes. The node B_1 has w children, each of which is a P_{i-1}-tree. A $\bar{P}_0$-tree is defined to be a full[6] tree

of depth one.

□

It is easy to see the depth of a $\bar{P}_i$-tree is

$$\text{depth}(\bar{P}_i) = 2i + 1. \tag{3.21}$$

The $\bar{P}_i$-trees for i = 0 and i = 1 (w = 3) are illustrated in Figure 3.4, and the $\bar{P}_i$-trees for i = 2 (w = 3) is illustrated in Figure 3.5.

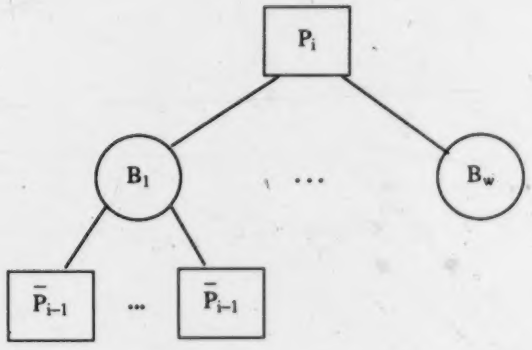


Figure 3.3. $\bar{P}_i$-tree.

Definition 3.10 describes the $\bar{P}_i$-tree as a specific collection of w $\bar{P}_{i-1}$-trees. This recursive definition is simple and especially useful for complexity considerations presented later in this section. Alternatively, the $\bar{P}_i$-tree can be described (less transparently) in terms of appropriate expansions of certain leaf nodes of a $\bar{P}_{i-1}$-tree.

Lemma 3.11. A $\bar{P}_i$-tree contains a $\bar{P}_{i-1}$-tree with the same root node.

Proof: For $i \geq 1$, it is asserted that the deletion of all nodes at depth 2i and 2i+1 from a $\bar{P}_i$-tree gives precisely a $\bar{P}_{i-1}$-tree . The result of the lemma then follows. The assertion is proved by induction.

---

6 A full tree [AHU83] of depth d is a tree in which every node at depth smaller than d has the same branching factor whereas all nodes at depth d are leaves. A full tree is termed a "uniform" tree in [Pear84].

For i = 1, clearly from Figure 3.4 that deletion of all nodes at depths 2 and 3 of a $\bar{P}_1$-tree gives a full tree of depth one, namely a $\bar{P}_0$-tree.

Assume that the assertion is true for a $\bar{P}_j$-tree , j = 1, ... , i-1, where i > 1. It is argued that the assertion then is also true for a $\bar{P}_i$-tree. From Figure 3.3, it is clear that the deletion of all nodes at depth 2i and 2i+1 of a $\bar{P}_i$-tree is equivalent to the deletion of all nodes of depths 2(i-1) and 2(i-1)+1 of the w $\bar{P}_{i-1}$-trees attached to B_1. However, by the inductive hypothesis, deletion of all nodes at depth 2(i-1) and 2(i-1)+1 of these w $\bar{P}_{i-1}$-trees converts them to w $\bar{P}_{i-2}$-trees, still attached to B_1. By Definition 3.10, this is precisely the $\bar{P}_{i-1}$-tree.

□

The expansion of certain leaves of the $\bar{P}_{i-1}$-tree to yield the $\bar{P}_i$-tree is, as shall be seen in Theorem 3.14 below, exactly the construction obtained by McAllester's algorithm initially in Stage I. It is necessary to know for which *i*, McAllester's algorithm successfully completes this stage. To this end, examine $\downarrow(\bar{P}_{i-1}, -\infty)$, the minimum number of conspirators required to decrease the minimax value of $\bar{P}_{i-1}$ to $-\infty$. In this direction, there is a stronger result.

Lemma 3.12. For any static evaluation function, let $\bar{T}$ rooted at a MAX node be a minimax tree such that

$$\bar{P}_{i-1} \subseteq \bar{T} \subset \bar{P}_i. \tag{3.22}$$

Then,

$$\downarrow(T, -\infty) = (i-1)(w-1)+w \tag{3.23}$$

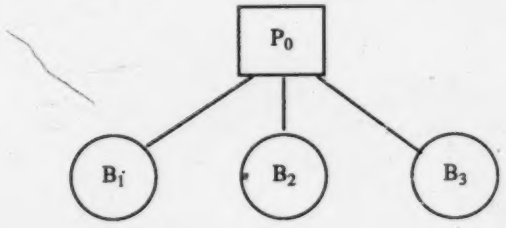
and

$$\uparrow(T, +\infty) = 1. \tag{3.24}$$

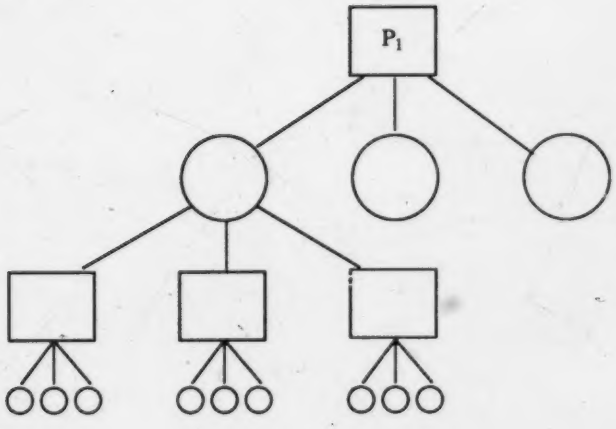
Proof: It is clear from (3.22) and Definition 3.10 that $\bar{T}$ contains (w - 1) leaf nodes at depth one. Any one of these leaves can conspire to make the minimax value of $\bar{T}$ become $+\infty$. Thus, (3.24) follows.

To prove (3.23), use induction on i. For the initial case, i = 1, it is required to show that if $\bar{T}$ is such that $\bar{P}_0 \subseteq \bar{T} \subset \bar{P}_1$, then $\downarrow(T, -\infty) = w$. There are two cases to consider, namely $\bar{T} = \bar{P}_0$ and $\bar{P}_0 \subset \bar{T} \subset \bar{P}_1$. In the

former case, $\bar{T}$ is a full tree of depth one; consequently, $\downarrow(T, -\infty) = \sum_{j=1}^{w} (1) = w$. In the latter case, $\bar{T}$ is



$\bar{P}_0$-tree for w = 3.



$\bar{P}_1$-tree for w = 3.

Figure 3.4. $\bar{P}_i$-trees.

described by means of Figure 3.6, where $\bar{T}_j$, j = 1,...,w is either a leaf or a full tree of depth one (compare $\bar{P}_0$ and $\bar{P}_1$ in Figure 3.4 and select $\bar{T}$ such that $\bar{P}_0 \subset \bar{T} \subset \bar{P}_1$). Furthermore, $\bar{T}_j$ is a leaf for at least one j; let k be such that $\bar{T}_k$ is a leaf. Then, $\downarrow(T_k, -\infty) = 1$ and

$$\downarrow(B_1,-\infty) = \min_{1\le j\le w} \{\downarrow(T_j,-\infty)\} = 1. \tag{3.25}$$

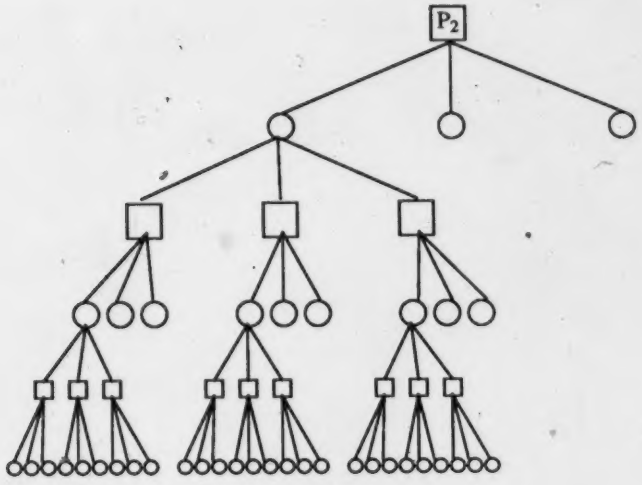


Figure 3.5. $\bar{P}_2$-tree for w = 3.

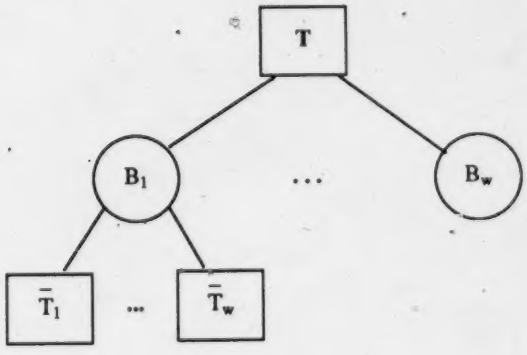


Figure 3.6. An intermediate $\bar{P}_1$-tree.

Consequently,

$$\downarrow(T,-\infty)) = \sum_{j=1}^{w} \downarrow(B_j,-\infty) = \sum_{j=1}^{w} (1) = w. \tag{3.26}$$

Hence, the initial case, i = 1, is proved.

Inductively, assume that for any tree $\bar{T}$ such that

$$\bar{P}_{i-1} \subseteq \bar{T} \subset \bar{P}_i, \tag{3.27}$$

the relationship

$$\downarrow(T,-\infty)=(i-1)(w-1)+w \tag{3.28}$$

holds. It will be shown that if $\bar{T}$ is any tree such that

$$\bar{P}_i \subseteq \bar{T} \subset \bar{P}_{i+1}, \tag{3.29}$$

then

$$\downarrow(T,-\infty)=i(w-1)+w. \tag{3.30}$$

Observe that a tree $\bar{T}$ satisfying (3.29) is given again by Figure 3.6, where $\bar{T}_j$ now satisfies

$$\bar{P}_{i-1} \subseteq \bar{T}_j \subseteq \bar{P}_i, \; j = 1, \ldots, w, \tag{3.31}$$

and where in addition for some k, $1 \le k \le w$,

$$\bar{P}_{i-1} \subseteq \bar{T}_k \subset \bar{P}_i. \tag{3.32}$$

From (3.31), (3.32), and the inductive hypotheses (3.27) and (3.28), it follows that

$$\downarrow(T_j,-\infty) \ge (i-1)(w-1)+w, \; 1 \le j \le w, \tag{3.33}$$

and

$$\downarrow(T_k,-\infty)=(i-1)(w-1)+w. \tag{3.34}$$

Thus,

$$\begin{aligned} \downarrow(B_1,-\infty) &= \min_{1 \le j \le w} \{\downarrow(T_j,-\infty)\} \\ &= (i-1)(w-1) + w, \end{aligned} \tag{3.35}$$

and

$$\begin{aligned} \downarrow(T,-\infty) &= \sum_{j=1}^{w} \downarrow(B_j,-\infty) \\ &= [(i-1)(w-1) + w] + \sum_{j=2}^{w} \downarrow(B_j,-\infty) \\ &= [(i-1)(w-1) + w] + w - 1 \end{aligned} \tag{3.36}$$

$$=i(w-1)+w.$$

□

Corollary 3.13 (dual): For any static evaluation function, let $\bar{T}$ rooted at a MIN node be a minimax tree such that

$$\bar{P}_{i-1} \subseteq \bar{T} \subset \bar{P}_i. \tag{3.37}$$

Then,

$$\uparrow(T,+\infty)=(i-1)(w-1)+w \tag{3.38}$$

and

$$\downarrow(T,-\infty)=1. \tag{3.39}$$

□

Theorem 3.14: In Stage I, McAllester's algorithm builds $\bar{P}_m$, where m is given by

$$m = \left\lceil \frac{CT-w}{w-1} \right\rceil. \tag{3.40}$$

The range of likely values for $\bar{P}_m$ is [e, +∞] for some finite e.

Proof: First, note that since it was assumed that CT > 1, therefore w - CT > w - 1. Thus, $\frac{CT-w}{w-1} > -1$ and m ≥ 0.

Note that since any finite minimax value of the root of a tree is equidistant from -∞ and +∞, McAllester chooses to eliminate -∞ from the range of likely values first; that is, in Stage I, the strategy selected in step M2 of McAllester's algorithm is always DecreaseRoot.

Observe that a tree consisting of a single node (the root node) cannot be convergent for CT > 1. Consequently, McAllester's algorithm begins (see step D2 of the algorithm) by expanding the root node, yielding $\bar{P}_0$. From (3.23) and (3.24), $\downarrow(P_0,-\infty)$ = w and $\uparrow(P_0, +\infty)$ =1. Thus, if CT ≤ w, then the range of likely values for $\bar{P}_0$ is [e, +∞]. The theorem is therefore true when CT ≤ w; equation (3.40) specifies m = 0.

When CT > w, the range of likely values remains $[-\infty,+\infty]$ and step M3 of McAllester's algorithm expands $\bar{P}_0$. It is shown that McAllester's algorithm successively builds the sequence $\bar{P}_0, \bar{P}_1, ..., \bar{P}_m$. Assert that given $\bar{P}_{j-1}$ for some j, $1 \le j \le m$, McAllester's algorithm next performs a number of expansions yielding $\bar{P}_j$, eventually. More specifically (and more strongly) assert that given *any* tree $\bar{T}$ such that

$$\bar{P}_{j-1} \subset \bar{T} \subset \bar{P}_j$$

for some j, $1 \le j \le m$, further expansion by McAllester's algorithm yields a tree $\bar{T}'$ such that

$$\bar{P}_{j-1} \subset \bar{T} \subset \bar{T}' \subseteq \bar{P}_j.$$

Since both $\bar{P}_{j-1}$ and $\bar{P}_j$ are finite, it follows that McAllester's algorithm eventually builds $\bar{P}_j$ from $\bar{P}_{j-1}$.

Proceed by induction. Assume that the assertion is true for j=1,...,i, where i < m. It is shown that the assertion is true for j=i+1; that is, it is shown that if $\bar{T}$ is such that

$$\bar{P}_i \subset \bar{T} \subset \bar{P}_{i+1}, \tag{3.41}$$

then McAllester's algorithm expands $\bar{T}$ to yield $\bar{T}'$ such that

$$\bar{P}_i \subset \bar{T} \subset \bar{T}' \subseteq \bar{P}_{i+1}. \tag{3.42}$$

For a tree $\bar{T}$ satisfying (3.41), from Lemma 3.12,

$$\downarrow(T,-\infty) = i(w-1) + w$$

and

$$\uparrow(T,+\infty) = 1.$$

However, (3.40) together with the assumption that i<m implies that $i < \frac{CT-w}{w-1}$; that is,

$$i(w-1) + w < CT.$$

Thus, $\downarrow(T,-\infty) < CT$ and the range of likely values for $\bar{T}$ is $[-\infty,+\infty]$. Therefore, the algorithm is still in Stage I and the strategy selected in step M2 of McAllester's algorithm is DecreaseRoot.

Now, examine more closely how the DecreaseRoot strategy expands T. From Definition 3.10, a tree $\bar{T}$ satisfying (3.41) must be as in Figure 3.6, where

$$\bar{P}_{i-1} \subset \bar{T}_j \subset \bar{P}_i, \ 1 \le j \le w, \tag{3.43}$$

and

$$\bar{P}_{i-1} \subseteq \bar{T}_k \subset \bar{P}_i \tag{3.44}$$

for at least one k, $1 \le k \le w$. In step D3 of McAllester's algorithm, $M = \{B_i, 1 \le i \le w\}$ because the minimax value $b_i > -\infty$ for all i. Therefore, B_1 (the leftmost node) is selected for expansion; that is, $\bar{B}_1$ is the input tree for the next recursive call to DecreaseRoot (made from Step D4); note that for this next call, "type" is now Minnode.

Now examine the expansion of B_1 by the DecreaseRoot strategy. Since "type" is now Minnode, McAllester's algorithm proceeds directly to Step D4 and expands $\bar{T}_k$, where k is the smallest integer (leftmost T_j) such that

$$\downarrow(T_k, -\infty) = \min_{1 \le j \le w} \{\downarrow(T_j, -\infty)\}.$$

From Lemma 3.12, it follows that k is the smallest integer satisfying (3.44). By the inductive hypothesis, expansion of $\bar{T}_k$ for this k then yields a tree $\bar{T}_k'$ such that

$$\bar{P}_{i-1} \subseteq \bar{T}_k \subset \bar{T}_k' \subseteq \bar{P}_i. \tag{3.45}$$

Expansion of $\bar{T}_k$ into $\bar{T}_k'$ converts $\bar{T}$ in Figure 3.6 into a tree $\bar{T}'$. According to (3.43),(3.44), and (3.45), clearly $\bar{T}'$ satisfies (3.42), completing the inductive argument.

Note that in all trees $\bar{T}'$ satisfying (3.42), clearly $\uparrow(T', +\infty) = 1$. Hence, $+\infty$ remains in the range of likely values. Also, observe that for $\bar{T}' = \bar{P}_m$, $\downarrow(P_m, -\infty) = m(w - 1) + w$, by Lemma 3.12. Since $m = \left\lceil \frac{CT-w}{w-1} \right\rceil$, then $\downarrow(P_m, -\infty) \ge CT$. Therefore, the range of likely values for $\bar{P}_m$ is $[e, +\infty]$, where e is finite.

□

Theorem 3.14 may be stated in a different but equivalent way; namely, successive applications of the DecreaseRoot strategy to Root, a MAX node, will yield eventually a $\bar{P}_i$-tree for any i. The dual, of course, states that successive applications of the IncreaseRoot strategy to Root, a MIN node, will yield eventually a $\bar{P}_i$-tree for any i.

Another equivalent statement of Theorem 3.14 is the following: Given a tree $\bar{T} \subset \bar{P}_i$ for any fixed i, the application of the DecreaseRoot strategy to $\bar{T}$ whose root is a MAX node yields $\bar{T}'$ such that $\bar{T} \subset \bar{T}' \subseteq \bar{P}_i$.

It is the dual of this statement that is required in Theorem 3.19 below.

Corollary 3.15. Given a tree $\bar{T} \subset \bar{P}_i$ for any fixed *i*, the application of the IncreaseRoot strategy to $\bar{T}$ whose root is a MIN node yields $\bar{T}'$ such that $\bar{T} \subset \bar{T}' \subseteq \bar{P}_i$.

□

After building the $\bar{P}_m$-tree (with a range of likely values [e,+∞]) during Stage I, McAllester's algorithm continues; in its quest to build a convergent tree, McAllester's algorithm begins Stage II. It is important to observe that later expansions performed in Stage II do not reintroduce -∞ into the range of likely values.

Lemma 3.16. Let $\bar{T}$ be any tree with range of likely values [e, ∞] for some finite e. Then any expansion(s) of $\bar{T}$ will not have -∞ in the range of likely values.

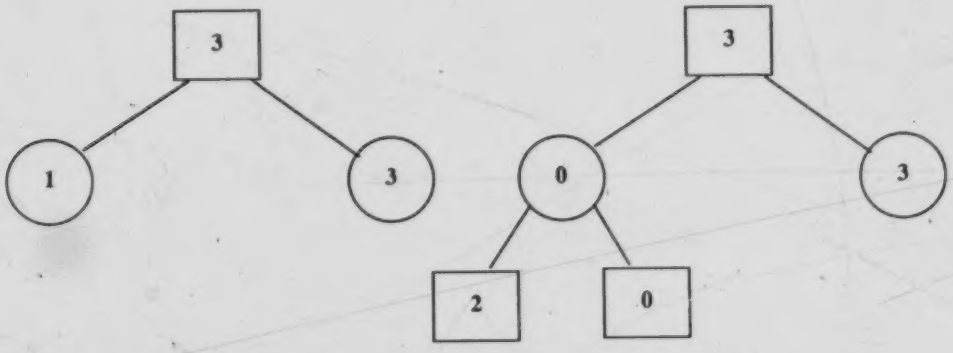
Proof: Since -∞ is not in the range of likely values for $\bar{T}$, there exists a set of leaves of $\bar{T}$ containing at least CT members such that no smaller set can make $\bar{T}$'s minimax value become -∞. It was assumed that the static evaluation function is finite. Thus, in later expansions, no leaf of $\bar{T}$ can become arbitrarily small. Hence, it will always require at least CT leaves to make the minimax value of the expanded tree become -∞. The lemma follows.

□

Note that for $\bar{T}$ with range of likely values [e, ∞], the lower bound, e, may change in the later expansion. For example, for the tree $\bar{T}$ in Figure 3.7, for CT = 2, the range of likely values of $\bar{T}$ changes from [1,∞] to [0,∞] on expansion of $\bar{T}$ to yield $\bar{T}'$.

Also, note that the range of likely values can increase when a tree $\bar{T}$ with a finite range of likely values is expanded. Consider the tree $\bar{T}$ and the expanded tree of Figure 3.8. For CT = 2, the range of likely values for $\bar{T}$ is [2,5], whereas the range of likely values for the expanded tree $\bar{T}'$ is [1,5]. Since the static value of a leaf node became 1, it required one fewer leaf node to conspire in the expanded tree for the root's minimax value to become 1.

The study of Stage II of McAllester's algorithm is now presented. Note in summary that Stage I involved successive applications of the DecreaseRoot strategy only, yielding eventually the $\bar{P}_m$-tree for which the range of likely values is [e, +∞]. Having removed -∞ but not +∞ from the range of likely values, using Lemma 3.16, it is clear from Step M2 of McAllester's algorithm that later expansions of the $\bar{P}_m$-tree involve only the IncreaseRoot strategy as long as +∞ remains in the range. It will be shown that successive applications of the IncreaseRoot strategy to the $\bar{P}_m$-tree yield eventually a new tree, the $\bar{C}_m$-tree, defined below.



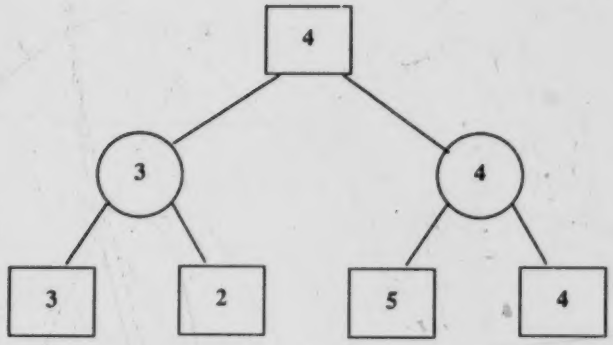
The tree $\bar{T}$:[1, ∞] Expanded tree $\bar{T}$:[0, ∞]

Figure 3.7. Lower bound of range of likely values changes.

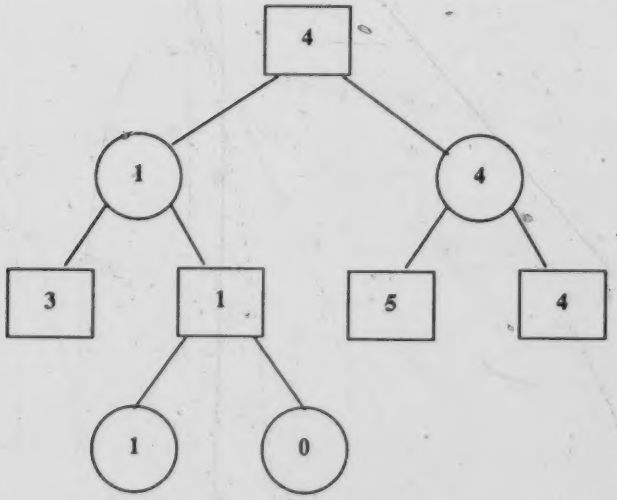
Definition 3.17: A $\bar{C}_i$-tree is defined recursively using Figure 3.9, where C_i is the root node. A $\bar{C}_0$-tree is defined to be a full tree of depth two.

□

It is shown that it is precisely when McAllister's algorithm completes building the $\bar{C}_m$-tree that the range of likely values becomes [b,d] for finite b and d. That is, Stage II of McAllester's algorithm involves the construction of the $\bar{C}_m$-tree from the $\bar{P}_m$-tree.



(a) the tree $\bar{T}$



(b) Expanded tree $\bar{T}$

Figure 3.8. Range of likely values increases after expansion.

Before proving that McAllester's algorithm builds a $\bar{C}_m$-tree, one further result is required.

Lemma 3.18. If $\bar{T}$ is such that

$$\bar{P}_i \subseteq \bar{T} \subseteq \bar{C}_i, \tag{3.46}$$

then

$$\uparrow(T,+\infty) \le i(w-1) + w, \tag{3.47}$$

with equality only if $\bar{T} = \bar{C}_i$.

Proof: The proof is by induction. For the initial case, it is required to show that if

$$\bar{P}_0 \subseteq \bar{T} \subseteq \bar{C}_0, \tag{3.48}$$

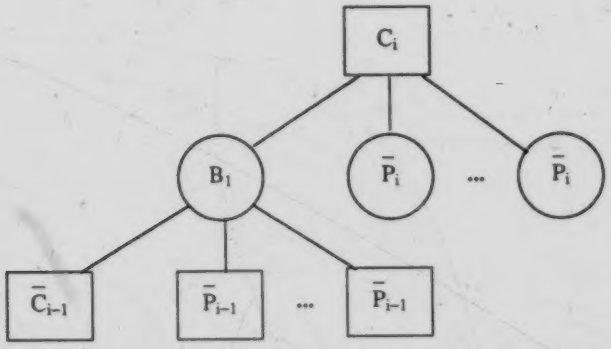


Figure 3.9. $\bar{C}_i$-tree.

then $\uparrow(T, +\infty) \le w$ with equality only if $\bar{T} = \bar{C}_0$. A tree $\bar{T}$ satisfying (3.48) can be represented by the tree shown in Figure 3.10, where $\bar{T}_k$, $1 \le k \le w$, is a leaf node or a full tree of depth one.

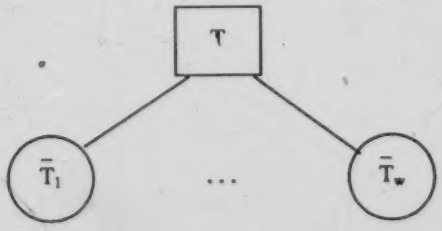


Figure 3.10. Intermediate $\bar{C}_0$-tree.

Thus,

$$\uparrow(T_k,+\infty) \le w, k = 1, \cdots, w, \tag{3.49}$$

with equality only if $\bar{T}_k$ is a full tree of depth one. Then

$$\uparrow(T,+\infty) = \min_{1\le k\le w} \{\uparrow(T_k,+\infty)\} \le w \tag{3.50}$$

with equality only if $\bar{T}_k$, for all k, is a full tree of depth one (i.e. with equality only if $\bar{T} = \bar{C}_0$).

Inductively, assume that if

$$\bar{P}_{i-1} \subseteq \bar{T} \subseteq \bar{C}_{i-1}, \tag{3.51}$$

then

$$\uparrow(T,+\infty) \le (i-1)(w-1) + w \tag{3.52}$$

with equality only if $\bar{T} = \bar{C}_{i-1}$. It is shown that for a tree satisfying (3.46) the relationship (3.47) holds. A tree satisfying (3.46) can be represented by Figure 3.11, where

$$\bar{P}_{i-1} \subseteq \bar{T}_1 \subseteq \bar{C}_{i-1}, \tag{3.53}$$

and

$$B_k \subseteq \bar{T}_k \subseteq \bar{P}_i \,, 2 \le k \le w \,. \tag{3.54}$$

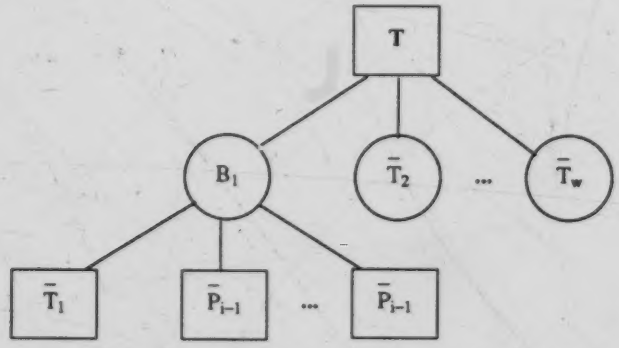


Figure 3.11. Intermediate $\bar{C}_{i-1}$-tree.

By the inductive hypotheses (3.51) and (3.52),

$$\uparrow(T_1,+\infty) \le (i-1)(w-1) + w \tag{3.55}$$

with equality only if $\bar{T}_1 = \bar{C}_{i-1}$. Thus,

$$\uparrow(B_1,+\infty) = \uparrow(T_1,+\infty) + (w-1)\uparrow(P_{i-1},+\infty) \tag{3.56}$$
$$\leq [(i-1)(w-1)+w] + (w-1)$$
$$= i(w-1) + w$$

with equality only if $\bar{T}_1 = \bar{C}_{i-1}$. Then from 3.56 and Corollary 3.13,

$$\uparrow(T,+\infty) = \min\{\uparrow(B_1,+\infty), \min_{2\leq k\leq w}\{\uparrow(T_k,+\infty)\}\}$$
$$\leq i(w-1) + w$$

with equality only if $\bar{T}_1 = \bar{C}_{i-1}$ and $\bar{T}_k = \bar{P}_i$, $k = 2, \ldots, w$ (i.e., with equality only if $\bar{T} = \bar{C}_i$).

□

Finally, it is shown that McAllester's algorithm builds a $\bar{C}_m$-tree.

Theorem 3.19. In Stage II, McAllester's algorithm builds $\bar{C}_m$ from $\bar{P}_m$, where m is given by (3.40). The range of likely values for the root of $\bar{C}_m$ is [b, d] for some finite b and d.

Proof: For the $\bar{P}_m$-tree obtained in stage I of McAllester's algorithm, according to Theorem 3.14, the range of likely values is [e, +∞] for some finite e. Because of Lemma 3.16, in Step M2 McAllester's algorithm continues to apply successively the IncreaseRoot strategy while +∞ remains in the range of likely values. It is shown that repeated applications of the IncreaseRoot strategy to the $\bar{P}_m$-tree eventually yield the $\bar{C}_m$-tree. The theorem then follows from Lemma 3.18 since +∞ is in the range of likely values for any tree $\bar{T}$ such that $\bar{P}_m \subseteq \bar{T} \subset \bar{C}_m$, but not for $\bar{C}_m$.

It is now shown, using induction, that successive applications of the IncreaseRoot strategy to $\bar{P}_j$, $j \geq 0$, yield $\bar{C}_j$. In particular, it is shown that this result is true for j = m, from which the validity of the theorem then follows. It is asserted that given $\bar{P}_j$, successive calls to IncreaseRoot yield $\bar{C}_j$ eventually. More specifically (and more strongly) it is asserted that given any tree $\bar{T}$ such that

$$\bar{P}_j \subseteq \bar{T} \subset \bar{C}_j \tag{3.57}$$

for some j, the next call to IncreaseRoot yields a tree $\bar{T}'$ such that

$$\bar{P}_j \subseteq \bar{T} \subset \bar{T}' \subseteq \bar{C}_j. \tag{3.58}$$

Initially, for j = 0, a tree $\bar{T}$ satisfying (3.57) can be represented by Figure 3.10, where $\bar{T}_k$ is either a leaf node or a full tree of depth one. It must be shown that the IncreaseRoot strategy applied to $\bar{T}$ yields a tree $\bar{T}'$ satisfying (3.58) with j = 0. With $\bar{T}$ so defined, the IncreaseRoot strategy proceeds without effect until it reaches Step I4 (because T is a MAX node and not a leaf). In Step I4, the minimal set M is the set of all $\bar{T}_k$ which are leaf nodes; this is true because $\uparrow(\bar{T}_k, +\infty) = 1$ when $\bar{T}_k$ is a leaf and $\uparrow(\bar{T}_k, +\infty) = w$ when $\bar{T}_k$ is a full tree of depth one. Consequently, also in Step I4, the leftmost leaf from the set M is expanded. This yields $\bar{T}'$ satisfying (3.58).

Inductively, assume that the theorem is true for j = 0,1, ... , i-1. It is shown that it is true for j = i; that is, it is shown that if $\bar{T}$ is such that

$$\bar{P}_i \subseteq \bar{T} \subset \bar{C}_i, \tag{3.59}$$

then IncreaseRoot expands $\bar{T}$ to yield $\bar{T}'$ such that

$$\bar{P}_i \subseteq \bar{T} \subset \bar{T}' \subseteq \bar{C}_i. \tag{3.60}$$

For a tree $\bar{T}$ satisfying (3.59), comparing $\bar{P}_i$ and $\bar{C}_i$ (see Figures 3.3 and 3.9), $\bar{T}$ must be the tree shown in Figure 3.11, where

$$\bar{P}_{i-1} \subseteq \bar{T}_1 \subseteq \bar{C}_{i-1} \tag{3.61}$$

and

$$B_k \subseteq \bar{T}_k \subseteq \bar{P}_i, \ 2 \le k \le w. \tag{3.62}$$

In addition, either

$$\bar{T}_1 \subset \bar{C}_{i-1}, \text{ or } \bar{T}_k \subset \bar{P}_i \text{ for some } k, \ 2 \le k \le w. \tag{3.63}$$

With $\bar{T}$ so defined, the IncreaseRoot strategy proceeds without effect until it reaches Step I4, as in the initial case. In Step I4, it must choose to expand one of $\bar{T}_k$, k = 2, ... , w, or the tree $\bar{B}_1$; the tree chosen is the one requiring the fewest conspirators to increase its minimax value to $+\infty$ (in case of a tie, the leftmost is chosen). To determine which of these trees to expand, from (3.62) and Corollary 3.13 observe that, for $2 \le k \le w$,

$$\uparrow(T_k,+\infty) \le i(w-1) + w$$

with equality if and only if $\bar{T}_k = \bar{P}_i$. For the remaining child, B_1, from (3.24), (3.61), and Lemma 3.18, observe that

$$\begin{aligned}\uparrow(B_1,+\infty) &= \uparrow(T_1,+\infty) + (w-1)\uparrow(P_{i-1},+\infty) \\ &\le [(i-1)(w-1) + w] + (w-1) \\ &= i(w-1) + w\end{aligned}$$

with equality if and only if $\bar{T}_1 = \bar{C}_{i-1}$. Thus, in Step I4, the algorithm selects to expand one of $\bar{T}_k$, $k = 2, \ldots, w$, or $\bar{B}_1$ for which the minimum number of conspirators is *strictly less than* $i(w-1) + w$; otherwise, $\bar{T}_k = \bar{P}_i$, $k = 2, \ldots, w$, and $\bar{T}_1 = \bar{C}_{i-1}$ which dictates that $\bar{T} = \bar{C}_i$ violating the inductive assumption (3.63).

In Step I4, if the subtree selected for expansion is $\bar{T}_k$ for some k, $2 \le k \le w$ (note that such a $\bar{T}_k$ must satisfy $\bar{T}_k \subset \bar{P}_i$), then the recursive call with input $\bar{T}_k$ of the IncreaseRoot strategy in Step I4 yields a tree $\bar{T}_k'$ such that $\bar{T}_k \subset \bar{T}_k' \subseteq \bar{P}_i$, according to Corollary 3.15. Clearly, such an expansion of the subtree $\bar{T}_k$ to yield $\bar{T}_k'$ gives a tree $\bar{T}'$ satisfying (3.60).

In Step I4, if the subtree selected for expansion is $\bar{B}_1$ (note that this can happen only if $\bar{T}_1 \subset \bar{C}_{i-1}$) examine the effect of the recursive call in Step I4 of IncreaseRoot with input $\bar{B}_1$ and type = Minnode. In this recursive call of IncreaseRoot with input $\bar{B}_1$, the first step of consequence is I3, since now type = Minnode. In Step I3, the child of $\bar{B}_1$ selected for expansion is $\bar{T}_1$. Since $\bar{T}_1$ is assumed to satisfy

$$\bar{P}_{i-1} \subseteq \bar{T}_1 \subset \bar{C}_{i-1},$$

the expansion of $\bar{T}_1$ (in the next recursive call of IncreaseRoot in Step I3 yields a tree $\bar{T}_1'$ satisfying

$$\bar{P}_{i-1} \subseteq \bar{T}_1 \subset \bar{T}_1' \subseteq \bar{C}_{i-1},$$

by the inductive hypothesis. Clearly, this expansion gives a tree $\bar{T}'$ satisfying (3.60).

□

There are several important consequences arising from Theorem 3.19. The first of these is given in the next corollary.

Corollary 3.20. For a constant static evaluation function, McAllester's algorithm converges. For given

CT, the uniform-McAllester tree is the $\bar{C}_m$-tree where m is given by (3.40).

Proof: First, consider any tree $\bar{T}$ produced by McAllester's algorithm such that $\bar{T} \subset \bar{C}_m$. From Lemma 3.18,

$$\uparrow(T, +\infty) < m(w-1) + w. \tag{3.64}$$

Using (3.64) and (3.40), $\bar{T}$ is not convergent; that is, the smallest tree produced by McAllester's algorithm that could possibly converge is the $\bar{C}_m$-tree.

It is now shown that for constant static evaluation function, the $\bar{C}_m$-tree does converge. Theorem 3.19 implies that for any static evaluation function (including a constant static evaluation function), McAllester's algorithm builds a $\bar{C}_m$-tree from a $\bar{P}_m$-tree.

Now, $\downarrow(\bar{C}_m, -\infty)$ is required. Lemma 3.12 implies that for a $\bar{P}_m$-tree,

$$\downarrow(P_m, -\infty) = m(w-1) + w. \tag{3.65}$$

Using (3.40), Lemma 3.16, and (3.65),

$$\downarrow(C_m, -\infty) \geq CT. \tag{3.66}$$

Next, obtain $\uparrow(C_m, +\infty)$. From Lemma 3.18

$$\uparrow(C_m, +\infty) = m(w-1) + w. \tag{3.67}$$

From (3.40) and (3.67)

$$\uparrow(C_m, +\infty) \geq CT. \tag{3.68}$$

At last, it can be shown that, for a constant static evaluation function, the $\bar{C}_m$-tree converges (the uniform-McAllester tree converges). For a constant static evaluation function, for any value $v < c_m$, using (3.66) implies that

$$\downarrow(C_m, v) \geq CT \tag{3.69}$$

(simply change the value of the leaves of the minimal set to v instead of to $-\infty$). Similarly, from (3.68), for any $v > c_m$,

$$\uparrow(C_m, v) \geq CT \tag{3.70}$$

Thus, the uniform-McAllester tree converges.

□

Thus, when the static evaluation function is constant, McAllester's algorithm is successful; on completion of Stage II, it terminates having built a convergent tree. When the static evaluation function is arbitrary, McAllester's algorithm proceeds to Stage III. In Stage III, McAllester's algorithm continues expanding the $\bar{C}_m$-tree. It is important to note that *any* tree built by McAllester's algorithm, *whether convergent or not*, must contain a $\bar{C}_m$-tree. Relative to the complexity diagram depicted in Figure 3.1, the following corollary has been proven.

Corollary 3.21. For given CT, the number of nodes in a McAllester tree is bounded below by the number of nodes in the uniform-McAllester tree (for the same CT).

□

Now the uniform-McAllester tree will be analyzed in terms of depth and number of nodes. These provide lower bounds for the depth and number of nodes in the McAllester tree. These lower bounds shall be given in terms of w and CT only. In preparation for this, it is necessary to obtain the depth of the uniform-McAllester tree in terms of m.

Lemma 3.22: The depth of a $\bar{C}_m$-tree, depth($\bar{C}_m$), is given by

$$\text{depth}(\bar{C}_m) = 2m+2. \tag{3.71}$$

Proof: First observe that the depth of a $\bar{P}_m$-tree is

$$\text{depth}(\bar{P}_m) = 2m+1, \tag{3.72}$$

since, by Definition 3.10, depth($\bar{P}_0$)= 1 and depth($\bar{P}_{m+1}$) = depth($\bar{P}_m$) + 2.

The result (3.71) is now shown to hold using induction. Initially, depth($\bar{C}_0$) = 2, since $\bar{C}_0$ is a full tree of depth two. Inductively, assume that (3.71) holds for a $\bar{C}_m$-tree. Then, for a $\bar{C}_{m+1}$-tree, from (3.72) and Definition 3.17, it follows that

$$\begin{aligned}\text{depth}(\bar{C}_{m+1}) &= \max\ \{(\text{depth}(\bar{C}_m)+2,\ \text{depth}(\bar{P}_m)+2,\ \text{depth}(\bar{P}_{m+1})+1\} \\ &= \max\ \{2m+4,\ 2m+3,\ 2m+4\}\end{aligned}$$

$=2m+4.$

□

Now obtain the depth of the uniform-McAllester tree.

Theorem 3.23. The depth, $depth_{CT}$, of a uniform-McAllester tree with conspiracy threshold CT is given by

$$depth_{CT} = 2\left\lceil \frac{CT-w}{w-1} \right\rceil + 2. \tag{3.73}$$

Proof: The result (3.73) is an immediate consequence of (3.40) and (3.71).

□

Now derive the formulae to determine how many nodes a uniform-McAllester tree will expand for given w and CT. For this purpose, the number of nodes in a $\bar{P}_m$-tree is required.

Lemma 3.24: The number of nodes in a $\bar{P}_m$-tree is

$$\#(\bar{P}_m) = \frac{(w^{m+1}-1)(w+1)}{w-1} \tag{3.74}$$

Proof: Since a $\bar{P}_0$-tree is a full tree of depth one, then $\#(\bar{P}_0) = w + 1$. Inductively, assume that (3.74) holds for a $\bar{P}_m$-tree. Then, from (3.74) and Definition 3.10, it follows that

$$\begin{aligned}\#(\bar{P}_{m+1}) &= w[\#(\bar{P}_m)]+w+1 \\ &= \frac{w(w^{m+1}-1)(w+1)}{w-1}+w+1 \\ &= \frac{(w^{m+2}-1)(w+1)}{w-1}\end{aligned}$$

□

To determine how many nodes there are in a uniform-McAllester tree, for given w and CT, the number of nodes in a $\bar{C}_m$-tree is required. For this purpose, the next lemma is presented.

Lemma 3.25. The number of nodes in a $\bar{C}_m$-tree is

$$\#(\bar{C}_m) = \frac{(w+1)^2 w^{m+1}-4}{w-1} - 2(m+1)w - 3. \tag{3.75}$$

Proof: Since a $\bar{C}_0$-tree is a full tree of depth two, clearly $\#(\bar{C}_0) = w^2 + w + 1$. Inductively, assume that (3.75) holds for a $\bar{C}_m$-tree. Then, for a $\bar{C}_{m+1}$-tree, using (3.74), (3.75) and Definition 3.17, it follows that

$$\begin{aligned}
\#(\bar{C}_{m+1}) &= \#(\bar{C}_m) + 2 + (w-1)[\#(\bar{P}_{m+1}) + \#(\bar{P}_m)] \\
&= \frac{(w+1)^2 w^{m+1} - 4}{w-1} - 2(m+1)w - 1 + (w-1)\left\{ \frac{(w^{m+2}-1)(w+1)}{w-1} + \frac{(w^{m+1}-1)(w+1)}{w-1} \right\} \\
&= \frac{(w+1)^2 w^{m+2} - 4}{w-1} - 2(m+2)w - 3.
\end{aligned}$$

□

Theorem 3.26. For given CT, the number of nodes in a uniform-McAllester tree is

$$N_{CT} = \frac{(w+1)^2 w^{\left\lfloor \frac{CT-w}{w-1} \right\rfloor + 1} - 4}{w-1} - 2\left\{ \left\lfloor \frac{CT-w}{w-1} \right\rfloor + 1 \right\} w - 3 \qquad (3.76)$$

Proof: The result (3.76) is an immediate consequence of (3.40) and (3.75).

□

The results of Theorem 3.23 and Theorem 3.26 are illustrated by Tables 3.1 and 3.2 below. The results of the tables have been verified through experiments performed with an implementation of McAllester's algorithm by Norbert Klingbeil [Klin89].

Note that there is no attempt to compare the number of nodes with the number of nodes generated by the alpha–beta algorithm. The comparisons would not be appropriate because, when using Conspiracy Numbers, depth is not a parameter; it is merely the depth of the tree produced by McAllester's algorithm. As well, the trees presented here are those that result after Stages I and II only of the algorithm. For an arbitrary static evaluation function, McAllester's algorithm is not yet finished; the depth of the tree when convergence occurs (as well as the number of nodes) is unknown.

It is useful to study the growth rate of the uniform-McAllester tree. For this reason, the next corollary is introduced.

Corollary 3.27. The number of nodes N_{CT} in a uniform-McAllester tree grows exponentially with CT

according to

$$N_{CT} = O(\gamma^{CT}), \tag{3.77}$$

	w = 2		w = 3		w = 4		w = 5	
CT	d	N_{CT}	d	N_{CT}	d	N_{CT}	d	N_{CT}
2	2	7	2	13	2	21	2	31
3	4	21	2	13	2	21	2	31
4	6	53	4	55	2	21	2	31
5	8	121	4	55	4	113	2	31
6	10	261	6	193	4	113	4	201
7	12	545	6	193	4	113	4	201
8	14	1117	8	619	6	505	4	201
9	16	2265	8	619	6	505	4	201
10	18	4565	10	1909	6	505	6	1091

Table 3.1. Depth and number of nodes in Uniform-McAllester tree.

	w = 10		w = 20		w = 30		w = 40	
CT	d	N_{CT}	d	N_{CT}	d	N_{CT}	d	N_{CT}
10	2	111	2	421	2	931	2	1641
20	6	13381	2	421	2	931	2	1641
30	8	134361	4	9201	2	931	2	1641
40	10	1344341	6	185561	4	29701	2	1641
50	12	13444321	6	185561	4	29701	4	68801
60	14	134444301	8	3713521	6	894541	4	68801
70	16	1344444281	8	3713521	6	894541	4	68801
80	18	13444444261	10	74273481	6	894541	6	2758321
90	20	134444444241	10	74273481	8	26841481	6	2758321
100	22	1344444444221	12	1485473441	8	26841481	6	2758321

Table 3.2. Depth and number of nodes in Uniform-McAllester tree.

where

$$\gamma = w^{\frac{1}{w-1}} \tag{3.78}$$

is the growth factor.

Proof: From (3.76), with w fixed,

$$N_{CT} = O\left[w^{\frac{CT-w}{w-1}+1}\right]$$

$$= O\left[\left(w^{\frac{1}{w-1}}\right)^{CT-1}\right]$$

$$= O(\gamma^{CT}).$$

□

Corollary 3.27 shows that the number of nodes in a tree built by McAllester's algorithm, whether convergent or not, grows exponentially with CT. The growth factor given in (3.78) decreases as w increases; therefore, for a constant static evaluation function and given CT, the number of nodes decreases as w increases (see Table 3.2).

3.5. Optimality

It was shown in Corollary 3.20 that McAllester's algorithm always begins by building a $\bar{C}_m$-tree, which is, in fact, the uniform-McAllester tree. Now, the significance of this result is discussed and the question of optimality is raised; that is, for a given CT and a constant static evaluation function, is there a tree containing fewer nodes than a uniform-McAllester tree which converges?

Referring to Figure 3.1, Definitions 3.2 and 3.3 imply that a uniform-minimal tree contains no more nodes than a uniform-McAllester tree. It is shown that, in fact, the number of nodes in a uniform-minimal tree is strictly less than the number of nodes in a uniform-McAllester tree for most CT. For these CT and for a constant static evaluation function, a convergent tree that has fewer nodes than the uniform-McAllester tree is constructed. That is, it is shown that the uniform-McAllester tree is not optimal (i.e., not uniform-minimal) in most instances.

The tree constructed is a full tree. Some preliminary results are required.

Lemma 3.28: For a full tree, $\bar{F}_i$, of depth 2i, $i \geq 0$,

$$\uparrow(\bar{F}_i, +\infty) = w^i \tag{3.79}$$

and

$$\downarrow(\bar{F}_i, -\infty) = w^i. \tag{3.80}$$

Proof: The theorem will be proved using induction. For the initial case, $i = 0$, $\bar{F}_i$ is a leaf node; equations (3.79) and (3.80) follow trivially.

Inductively assume that if $\bar{F}_k$ is a full tree of depth 2k, then

$$\uparrow(\bar{F}_k, +\infty) = w^k \tag{3.81}$$

and

$$\downarrow(\bar{F}_k, -\infty) = w^k. \tag{3.82}$$

The full tree $\bar{F}_{k+1}$ of depth 2k+2 can be represented as in Figure 3.12.

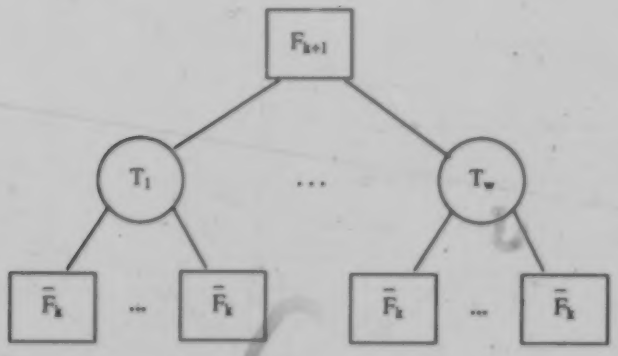


Figure 3.12. $\bar{F}_{k+1}$-tree.

Since T_j, $j = 1, \ldots, w$, is a MIN node, it follows from (3.81) that

$$\uparrow(T_j, +\infty) = \sum_{k=1}^{w} \uparrow(\bar{F}_k, +\infty) = w^{k+1} \tag{3.83}$$

and from (3.82) that

$$\downarrow(T_j, -\infty) = \min_{1 \leq k \leq w} [\downarrow(\bar{F}_k, -\infty)] = w^k. \tag{3.84}$$

Thus,

$$\uparrow(F_{k+1},+\infty) = \min_{1 \le j \le w} \{\uparrow(T_j,+\infty)\} = w^{k+1} \tag{3.85}$$

and

$$\downarrow(F_{k+1},-\infty) = \sum_{j=1}^{w} \downarrow(T_j,-\infty) = w^{k+1}. \tag{3.86}$$

□

To show that the uniform-McAllester tree is not optimal, compare the number of nodes in a uniform-McAllester tree with $\#(\bar{F}_i)$. Such a comparison would be meaningless without first establishing that both trees converge for a given CT. For the uniform-McAllester tree, this was established in Corollary 3.20. For the full tree, the following corollary is required.

Corollary 3.29: For CT ≤ w^j, and for a constant static evaluation function, the full tree $\bar{F}_i$ of depth 2i is convergent.

Proof: The proof is the same as that of the proof of Corollary 3.20.

□

Whereas the discussion in this section revolves around a constant static evaluation function, the following result for an arbitrary static evaluation function arises from Lemma 3.28 and is stated now because it will be required in Chapter 4.

Corollary 3.30: For CT ≤ w^j, and for an arbitrary static evaluation function, the range of likely values of the full tree $\bar{F}_i$, of depth 2i, is [b,d] for some finite b and d.

Proof: Let f_i be the minimax value of $\bar{F}_i$. Since $\uparrow(F_i,f_i) = \downarrow(F_i,f_i) = 0$, then (3.79), (3.80) and the monotonicity of ↑ and ↓[7] imply that the number of conspirators required to attain any finite value is finite.

---

7 As the minimax value increases above f_i, ↑ is non-decreasing and as the minimax value decreases below f_i, ↓ is also non-decreasing [Monotonicity Theorem, McA185].

For $CT \leq w^i$, the corollary follows.

□

To show that for a constant static evaluation function the uniform-McAllester tree is not optimal, it is shown that there exists some tree, namely the $\bar{F}_i$-tree for an appropriate i, with fewer nodes.

Lemma 3.31. The number of nodes in a full tree $\bar{F}_i$ of depth 2i is

$$\#(\bar{F}_i) = \frac{w^{2i+1}-1}{w-1}. \tag{3.87}$$

Proof: The proof uses induction. For the initial case, $\bar{F}_0$ is a leaf node and $\#(\bar{F}_0) = 1$. Inductively assume the theorem is true for i = 0, 1, ... , k; that is, assume

$$\#(\bar{F}_k) = \frac{w^{2k+1}-1}{w-1}. \tag{3.88}$$

Then, using (3.88), an examination of Figure 3.12 yields

$$\begin{aligned}\#(\bar{F}_{k+1}) &= w^2\,\#(\bar{F}_k) + w + 1 \\ &= w^2\frac{w^{2k+1}-1}{w-1} + w + 1 \\ &= \frac{w^{2k+3}-1}{w-1}\end{aligned}$$

□

For a given CT, for a constant static evaluation function, it is desired to show that McAllester's algorithm is not optimal. This is shown for certain CT, namely $CT = w^i$ (for any positive integer i), in the following theorem.

Theorem 3.32. For $CT = w^i$, $i \geq 1$, and for a constant static evaluation function, there exists a convergent tree with

$$\frac{(CT^2)w-1}{w-1} \tag{3.89}$$

nodes.

Proof: For $CT = w^i$, construct the $\bar{F}_i$-tree. The proof is then an immediate consequence of Corollary

3.29 and Lemma 3.31.

□

Now, it is possible to compare the number of nodes in the uniform-McAllester tree and $\bar{F}_i$ for certain specific CT. This represents the first step in comparing these two trees for all CT's.

Theorem 3.33: For $CT = w^i$, $i \geq 2$, and for a constant static evaluation function, there exists a convergent tree with strictly fewer nodes than a uniform-McAllester tree.

Proof: The uniform-McAllester tree is the $\bar{C}_m$-tree where for $CT = w^i$

$$m = \left\lceil \frac{CT-w}{w-1} \right\rceil = \frac{CT-w}{w-1}.$$

According to (3.76), the number of nodes in the $\bar{C}_m$-tree is given by

$$\#(\bar{C}_m) = \frac{(w+1)^2 w^{\frac{CT-1}{w-1}} - 2wCT - w - 1}{w-1}. \tag{3.90}$$

The result follows from Theorem 3.32 by comparing (3.89) with (3.90).

Note that for $CT = w$, (3.89) and (3.90) are equal.

□

Rephrasing Theorem 3.33,yields the following corollary.

Corollary 3.34. For $CT = w^i$, $i \geq 2$, the uniform-McAllester tree is not optimal.

□

Indeed, the uniform-McAllester tree is far from optimal. In Corollary 3.27, it was shown that the number of nodes in a uniform-McAllester tree grows exponentially with CT. For the special sequence of conspiracy thresholds $CT = w^i$, $i \geq 1$, however, according to (3.89) there exist convergent trees (for constant static evaluation function) for which the number of nodes grows quadratically with CT.

It is now known that, for $CT = w^i$, a convergent tree (the $\bar{F}_i$-tree) may be constructed with fewer nodes than the uniform-McAllester tree, except for $i = 1$ (where the number of nodes in the two trees is

equal). Next, compare the two trees in general. That is, look at what happens for $w^{i-1} < CT \le w^i$.

Recall that, for a given CT and for a constant static evaluation function, the intent is to find a convergent tree with fewer nodes than the uniform-McAllester tree. Theorem 3.33 achieves this goal for $CT = w^i$, $i \ge 2$. For an arbitrary CT (in particular for $w^{i-1} < CT < w^i$), the uniform McAllester tree is the $\bar{C}_m$-tree, where

$$m = \left\lfloor \frac{CT-w}{w-1} \right\rfloor.$$

From Corollary 3.29, it follows that the $\bar{F}_i$-tree, where

$$i = \left\lceil \log_w CT \right\rceil, \tag{3.91}$$

is also convergent for a constant static evaluation function. Comparing the number of nodes in these two trees yields the following theorem.

Theorem 3.35. For any $CT \ge w^3$, and for a constant static evaluation function, there exists a convergent tree with fewer nodes than the uniform-McAllester tree.

Proof: From (3.87), with i given by (3.91), the number of nodes in the $\bar{F}_i$-tree is given by

$$\#(\bar{F}_i) = \frac{w^{2\left\lceil \log_w CT \right\rceil + 1} - 1}{w-1}. \tag{3.92}$$

The result follows by comparing (3.92) with the number of nodes (3.76) in the $\bar{C}_m$-tree, for $CT \ge w^3$.

□

Corollary 3.36. For a constant static evaluation function, there exist convergent trees for which the number of nodes grows quadratically with CT.

Proof: The result is an immediate consequence of (3.92).

□

For $CT < w^3$, there are cases where the uniform-McAllester tree contains fewer nodes than the smallest convergent full tree.

Example 3.37. For CT = w+1, the uniform-McAllester tree is $\bar{C}_1$ since

$$m = \left\lceil \frac{CT-w}{w-1} \right\rceil = 1.$$

From (3.75), the number of nodes in the uniform-McAllester tree is

$$\#(\bar{C}_1) = w^2(w+3)+1.$$

On the other hand, the smallest convergent full tree is $\bar{F}_2$, since

$$i = \left\lceil \log_w(w+1) \right\rceil = 2.$$

From (3.79) and (3.80), note that $\bar{F}_1$ is not convergent. The number of nodes in $\bar{F}_2$, according to (3.87), is given by

$$\#(\bar{F}_2) = \frac{w^5-1}{w-1}.$$

Clearly, $\#(\bar{C}_1) < \#(\bar{F}_2)$ for all w.

□

In contrast to Example 3.37, for $CT < w^3$, there are cases where the smallest convergent contains fewer nodes than the uniform-McAllester tree.

Example 3.38: Let CT = 3w-2, where $w \geq 3$. Since

$$m = \left\lceil \frac{CT-w}{w-1} \right\rceil = 2,$$

the uniform-McAllester tree is $\bar{C}_2$, which according to (3.75) contains

$$\#(\bar{C}_2) = \frac{(w+1)^2w^3-4}{w-1} - 6w - 3$$

$$= w^4 + 3w^3 + 4w^2 - 2w + 1$$

nodes. On the other hand, the smallest convergent full tree is $\bar{F}_2$ (since $i = \left\lceil \log_w CT \right\rceil = 2$), which according to (3.87) contains

$$\#(\bar{F}_2) = \frac{w^5-1}{w-1}$$

$$= w^4 + w^3 + w^2 + w + 1$$

nodes. Thus, for $w \geq 3$, the full tree contains fewer nodes than the uniform McAllester tree.

However, it is possible to do better. Let $\bar{T}$ be given as in Figure 3.13, where T has children $T_1, \ldots, T_w$ and the children of $T_4, \ldots, T_w$ are the same as the children of T_3. Since $\uparrow(C_0, +\infty) = w$, $\downarrow(C_0, -\infty) = w$, $\uparrow(P_0, +\infty) = 1$, and $\downarrow(P_0, -\infty) = w$,

$$\uparrow(T_j, +\infty) = \begin{cases} 2\uparrow(C_0, +\infty) + (w-2)\uparrow P_0, +\infty) = 3w - 2, & j = 1,2, \\ \uparrow(T_j, +\infty) = 2\uparrow(C_0, +\infty) + (w-2) = 3w - 2, & j = 3, \cdots, w, \end{cases}$$

and

$$\downarrow(T_j, -\infty) = \begin{cases} \min\{\downarrow(C_0 - \infty), \downarrow(P_0, -\infty)\} = w, & j = 1,2, \\ \downarrow(T_j, -\infty) = \min\{\downarrow(C_0 - \infty), 1\} = 1, & j = 3, \cdots, w. \end{cases}$$

Thus,

$$\uparrow(T, +\infty) = \min_{1 \leq j \leq w} \{\uparrow(T_j, +\infty)\} = 3w - 2$$

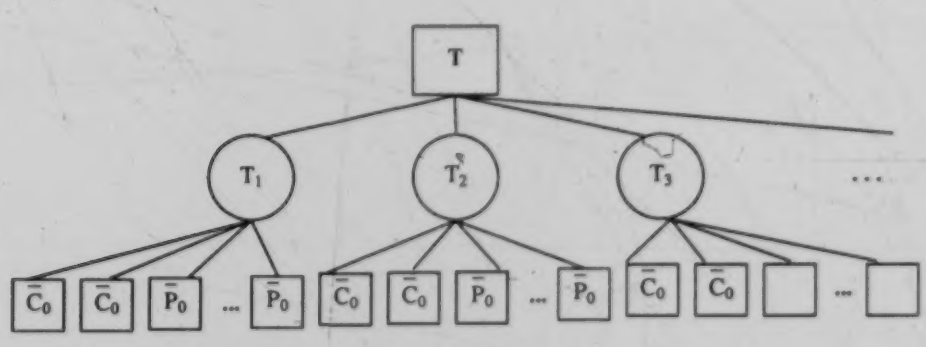


Figure 3.13. Intermediate full tree.

and

$$\downarrow(T, -\infty) = \sum_{j=1}^{w} \downarrow(T_j, -\infty) = 3w - 2$$

and consequently, $\bar{T}$ is also convergent. Clearly, the number of nodes in $\bar{T}$ is

$$\#(\bar{T}) = 2w^3 + 5w^2 - 3w + 1$$

which is smaller than #($\bar{F}_2$).

□

In both Example 3.37 and 3.38, CT is such that $w < CT \leq w^2$. For this CT, from (3.79) and (3.80), clearly the full tree $\bar{F}_2$ converges but $\bar{F}_1$ does not. In terms of full trees, it therefore makes sense only to compare $\bar{F}_2$ with the uniform-McAllester tree. However, it is possible that there is a tree between $\bar{F}_1$ and $\bar{F}_2$ containing fewer nodes than $\bar{F}_2$ that also converges. The two examples bring home the point. In Example 3.37, $\bar{C}_1$ is such that $\bar{F}_1 \subset \bar{C}_1 \subset \bar{F}_2$, and, in Example 3.38, $\bar{T}$ also satisfies $\bar{F}_1 \subset \bar{T} \subset \bar{F}_2$. It is believed that the two trees, $\bar{C}_1$ in Example 3.37 and $\bar{T}$ in Example 3.38, are optimal, that is, there exists no other convergent tree with fewer nodes. For an arbitrary CT, a stronger statement is made.

Conjecture 3.39. For any CT and for a constant static evaluation function, a convergent tree T with the fewest nodes satisfies

$$\bar{F}_{i-1} \subset \bar{T} \subseteq \bar{F}_i$$

where $i = \left\lceil \log_w CT \right\rceil$. In addition, for an arbitrary static evaluation function, this same tree is the tree with the fewest nodes which removes $-\infty$ and $+\infty$ from the range of likely values.

As a step in this direction the following theorem is presented.

Theorem 3.40. For $CT \leq w$, the full tree $\bar{F}_1$ of depth 2 is optimal.

Proof: Suppose $\bar{F}_1$ is not optimal. Then there exists some convergent tree $\bar{T}$ which has at least one leaf L at depth one. Since

$$\uparrow(L,+\infty) = 1,$$

then,

$$\uparrow(T,+\infty) = 1.$$

Hence, a contradiction exists; $\bar{T}$ is not convergent since it is assumed that $CT > 1$.

□

Chapter 4

Improving McAllester's Algorithm

4.1. Introduction

In Chapter 3, McAllester's algorithm was analyzed. In particular, it was shown that, in the process of eliminating $+\infty$ and $-\infty$ from the range of likely values, McAllester's algorithm always produces a $\overline{C}_m$-tree. Furthermore, it was shown that the number of nodes in the $\overline{C}_m$-tree grows exponentially with CT. McAllester's algorithm *always* grows a $\overline{C}_m$-tree (depth 2m+2) *before it even considers the given static evaluation function.* The C_m-tree is biased towards some nodes; some are expanded deeply whereas others get only a shallow expansion. The nodes are expanded in a preset manner. Thus, for any given static evaluation function, McAllester's algorithm expands an exponential number of nodes in a preset pattern.

This chapter presents an improvement to McAllester's algorithm; as in McAllester's algorithm, the improved algorithm CNEW begins by expanding a tree in a preset pattern. CNEW differs from McAllester's algorithm in one major respect; in the first stages of the algorithm that remove $-\infty$ and $+\infty$ from the range of likely values, it produces a shallower tree and in the process expands only a modest number of nodes; instead of exponentially, the number of nodes grows only quadratically with CT.

4.2. The Improved Algorithm

The new algorithm was designed after the analysis of the full tree was done in Chapter 3.5. This analysis suggested that a tree with more breadth than the $\overline{C}_m$-tree would enhance the performance of McAllester's algorithm. Adding breadth to the tree produced is accomplished by the modification to McAllester's algorithm given in the following algorithm.

Algorithm 4.1 (CNEW):

Only two steps of McAllester's algorithm are altered.

Replace Step D3 of McAllester's algorithm by

D3': /* Non-leaf which is Maxnode */

If $\bar{T} \neq$ leaf and type=Maxnode

Then

M← $\{\bar{T}_i: t_i > t_{min}\}$ /* determine minimal set */

Select leftmost $\bar{T}_j$ **such that** $\downarrow(T_j, t_{min}) \leq \downarrow(T_i, t_{min})$ **for all** $\bar{T}_i$ **of M**

$[\bar{T}_j, t_j, \uparrow(T_j,v), \downarrow(T_j,v)]$← DecreaseRoot($\bar{T}_j$, t_{min}, Minnode)

End If

Replace Step I3 of McAllester's algorithm by

I3': /* Non-leaf which is Minnode */

If $\bar{T} \neq$ leaf and type=Minnode

Then

M← $\{\bar{T}_i: t_i < t_{max}\}$ /* determine minimal set */

Select leftmost $\bar{T}_j$ **such that** $\uparrow(T_j, t_{max}) \leq \uparrow(T_i, t_{max})$ **for all** $\bar{T}_i$ **of M**

$[\bar{T}_j, t_j, \uparrow(T_j,v), \downarrow(T_j,v)]$← IncreaseRoot($\bar{T}_j$, t_{max}, Maxnode)

End If

□

To analyze CNEW, further definitions and lemmas are required. Notation is required for full trees of odd depth (recall that $d(\bar{F}_i) = 2i$). Therefore, the following definition is presented.

Definition 4.2. $\bar{E}_j$, $j \geq 1$, is a full tree of depth $2j - 1$.

□

When CNEW is being analyzed, it is necessary to know which subtree will be expanded next at any particular instance. This, in turn, requires knowing the number of conspirators necessary to change the

minimax value of each subtree already produced by CNEW. For these calculations, the following lemmas are presented.

Lemma 4.3. Let $\bar{T}$ be a tree such that $\bar{E}_{j-1} \subset \bar{T} \subset \bar{E}_j$. Then

$$w^{j-1} \le \downarrow(T,-\infty) < w^j.$$

Proof: The proof uses induction on j. For brevity, the base step (j=2) is omitted. Inductively, assume the assertion is true for j = k; that is, assume that for a tree $\bar{T}$ such that $\bar{E}_{k-1} \subset \bar{T} \subset \bar{E}_k$,

$$w^{k-1} \le \downarrow(T,-\infty) < w^k. \tag{4.1}$$

It is shown that the assertion is true for j=k+1. Show that given any tree $\bar{T}$ such that $\bar{E}_k \subset \bar{T} \subset \bar{E}_{k+1}$,

$$w^k \le \downarrow(T,-\infty) < w^{k+1}. \tag{4.2}$$

Such a tree $\bar{T}$ is illustrated in Figure 4.1, where

$$\bar{F}_{k-1} \subset \bar{B}_p \subset \bar{F}_k, \text{ for all } p,\ 1 \le p \le w, \tag{4.3}$$

$$\text{and } \bar{F}_{k-1} \subset \bar{B}_q \subset \bar{F}_k, \text{ for some } q,\ 1 \le q \le w, \tag{4.4}$$

or, equivalently,

$$\bar{E}_{k-1} \subset \bar{T}_{p,r} \subset \bar{E}_k, \text{ for all } p,r\ 1 \le p,r \le w, \tag{4.5}$$

$$\text{and } \bar{E}_{k-1} \subset \bar{T}_{q,s} \subset \bar{E}_k, \text{ for some } q,s,\ 1 \le q,s \le w. \tag{4.6}$$

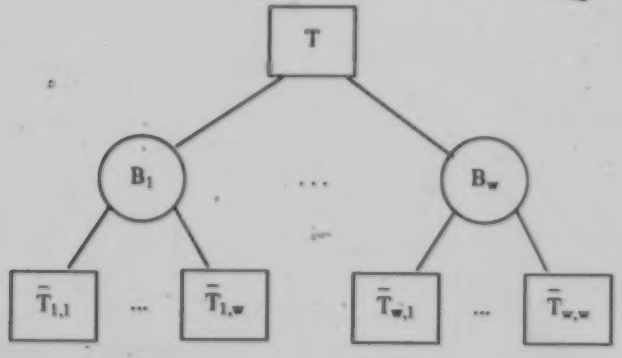


Figure 4.1. Tree.

If $\bar{B}_p = \bar{F}_k$, then (3.79) implies

$$\downarrow(B_p,-\infty) = w^k. \tag{4.7}$$

Since B_q is a MIN node,

$$\downarrow(B_q,-\infty) = \min_{1\le s\le w} \{\downarrow(T_{q,s},-\infty)\}.$$

When $\bar{B}_q \subset \bar{F}_k$,

$$\bar{E}_{k-1} \subseteq \bar{T}_{q,s} \subseteq \bar{E}_k, \text{ for all } q,s,\ 1 \le q,s \le w$$

and, for some s, (4.6) is true. Therefore, (4.1) implies

$$w^{k-1} \le \downarrow(B_q,-\infty) < w^k. \tag{4.8}$$

Since T is a MAX node,

$$\downarrow(T,-\infty) = \sum_{i=1}^{w} \downarrow(B_i,-\infty). \tag{4.9}$$

The assertion (4.2) now follows from (4.7),(4.8), and (4.9).

□

Lemma 4.4. Let $\bar{T}$ be a tree such that $\bar{E}_j \subseteq \bar{T} \subset \bar{F}_j$. Then

$$w^{j-1} \le \uparrow(T,+\infty) < w^j.$$

Proof: The proof uses induction on j. For brevity, the base step (j=1) is omitted. Inductively, assume the assertion is true for j = k; assume that for a tree $\bar{T}$ such that $\bar{E}_k \subseteq \bar{T} \subset \bar{F}_k$,

$$w^{k-1} \le \uparrow(T,+\infty) < w^k. \tag{4.10}$$

It is shown that the assertion is true for j=k+1. Show that given any tree $\bar{T}$ such that $\bar{E}_{k+1} \subseteq \bar{T} \subset \bar{F}_{k+1}$,

$$w^k \le \uparrow(T,+\infty) < w^{k+1}. \tag{4.11}$$

Such a tree $\bar{T}$ is illustrated in Figure 4.1, where now

$$\bar{F}_k \subset \bar{B}_p \subseteq \bar{E}_{k+1}, \text{ for all } p,\ 1 \le p \le w, \tag{4.12}$$

$$\text{and } \bar{F}_k \subseteq \bar{B}_q \subset \bar{E}_{k+1}, \text{ for some } q,\ 1 \le q \le w; \tag{4.13}$$

or, equivalently,

$$\bar{E}_k \subseteq \bar{T}_{p,r} \subset \bar{F}_k, \text{ for all } p,r,\ 1 \le p,r \le w, \tag{4.14}$$

$$\text{and } \bar{E}_k \subseteq \bar{T}_{q,s} \subset \bar{F}_k, \text{ for some } q,s,\ 1 \le q,s \le w. \tag{4.15}$$

If $\bar{B}_p = \bar{E}_{k+1}$, then using the methods applied in proving Lemma 3.28, it may be shown that since $\bar{B}_p$ is

rooted at a MIN node,

$$\uparrow(B_p,+\infty) = w^{k+1}. \tag{4.16}$$

When $\bar{B}_q \subset \bar{E}_{k+1}$,

$$\bar{E}_k \subset \bar{T}_{q,s} \subset \bar{F}_k, \text{ for all } q,s,\ 1 \le q,s \le w$$

and, for some s, (4.15) is true. For this $\bar{T}_{q,s}$, (4.10) implies

$$w^{k-1} \le \uparrow(T_{q,s},+\infty) < w^k. \tag{4.17}$$

Since $\bar{B}_q$ is rooted at a MIN node, summing yields

$$w^k \le \uparrow(B_q,+\infty) < w^{k+1}. \tag{4.18}$$

Since T is a MAX node,

$$\uparrow(T,+\infty) = \min_{1 \le i \le w} \uparrow(B_i,+\infty). \tag{4.19}$$

The assertion (4.11) now follows from (4.16), (4.17), (4.18) and (4.19).

□

Lemma 4.5. Let $\bar{F}_{j-1} \subseteq \bar{T} \subset \bar{F}_j$. Then

$$w^{j-1} \le \uparrow(T,+\infty) < w^j.$$

Proof: The proof uses induction on j. For brevity, the base step (j=2) is omitted. Inductively, assume the assertion is true for j = k; assume that for a tree $\bar{T}$ such that $\bar{F}_{k-1} \subseteq \bar{T} \subset \bar{F}_k$,

$$w^{k-1} \le \uparrow(T,+\infty) < w^k. \tag{4.20}$$

It is shown that the assertion is true for j=k+1. Show that given any tree $\bar{T}$ such that $\bar{F}_k \subseteq \bar{T} \subset \bar{F}_{k+1}$,

$$w^k \le \uparrow(T,+\infty) < w^{k+1}. \tag{4.21}$$

Such a tree $\bar{T}$ is illustrated in Figure 4.1, where now

$$\bar{E}_k \subset \bar{B}_p \subset \bar{E}_{k+1}, \text{ for all } p,\ 1 \le p \le w, \tag{4.22}$$

$$\text{and } \bar{E}_k \subset \bar{B}_q \subset \bar{E}_{k+1}, \text{ for some } q,\ 1 \le q \le w; \tag{4.23}$$

or, equivalently,

$$\bar{F}_{k-1} \subseteq \bar{T}_{p,r} \subseteq \bar{F}_k, \text{ for all } p,r,\ 1 \le p,r \le w, \tag{4.24}$$

$$\text{and } \bar{F}_{k-1} \subseteq \bar{T}_{q,s} \subset \bar{F}_k, \text{ for some } q,s,\ 1 \le q,s \le w. \tag{4.25}$$

If $\bar{B}_p = \bar{E}_{k+1}$, then in a similar way to that used in proving Lemma 3.28, it may be shown that

$$\uparrow(B_p,+\infty) = w^k. \tag{4.26}$$

As in the proof of Lemma 4.4, when (4.23) is true, (4.25) is true for some s, and (4.20) implies

$$w^k \le \uparrow(B_q,+\infty) < w^{k+1}. \tag{4.27}$$

Since T is a MAX node,

$$\uparrow(T,+\infty) = \min_{1\le i\le w} \uparrow(B_i,+\infty). \tag{4.28}$$

The inductive assumption (4.21) now follows from (4.26),(4.27), and (4.28).

□

The study of CNEW is presented in three stages. The first stage will cover the elimination of $-\infty$ from the range of likely values. The second stage eliminates $+\infty$ from the range of likely values. The third and final stage continues the algorithm after $+\infty$ has been eliminated from the range of likely values until convergence occurs.

To study the first stage, the following lemma is presented.

Lemma 4.6. If $\bar{T}$ is such that $\bar{E}_{j-1} \subseteq \bar{T} \subset \bar{E}_j$, then the DecreaseRoot strategy in CNEW yields $\bar{T}'$ such that $\bar{E}_{j-1} \subseteq \bar{T} \subset \bar{T}' \subseteq \bar{E}_j$.

Proof: The proof uses induction on j. For brevity, the base step (j=2) is omitted. Assume that the lemma is true for j=2,...,k. It is shown that the lemma is true for j=k+1. Show that given any tree $\bar{T}$ such that

$$\bar{E}_k \subseteq \bar{T} \subset \bar{E}_{k+1}, \tag{4.29}$$

CNEW expands $\bar{T}$ to yield $\bar{T}'$ such that

$$\bar{E}_k \subseteq \bar{T} \subset \bar{T}' \subseteq \bar{E}_{k+1}. \tag{4.30}$$

For a tree $\bar{T}$ satisfying (4.29), $\bar{T}$ must be the tree shown in Figure 4.1, where

$$\bar{E}_{k-1} \subseteq \bar{T}_{p,r} \subseteq \bar{E}_k, \text{ for all } p,r\ 1 \le p,r \le w, \tag{4.31}$$

$$\text{and } \bar{E}_{k-1} \subseteq \bar{T}_{q,s} \subset \bar{E}_k, \text{ for some } q, s,\ 1 \le q, s \le w. \tag{4.32}$$

If $\bar{T}_{p,r} = \bar{E}_k$, Lemma 4.3 implies that $w^k \le \downarrow(\bar{T}_{p,r}, -\infty) < w^{k+1}$. When $\bar{T}_{q,s} \subset \bar{E}_k$, Lemma 4.3 implies that $w^{k-1} \le \downarrow(\bar{T}_{q,s}, -\infty) < w^k$. Therefore, in the first step of consequence in DecreaseRoot, Step D3', M is the set of nodes B_i which have one or more children $\bar{T}_{i,s} \subset \bar{E}_k$. The leftmost of these is selected; denote this to be node B_q. The resulting subsequent call to DecreaseRoot proceeds without effect to Step D4. Now, M is the child(ren) $\bar{T}_{q,s}$ of B_q such that $\bar{T}_{q,s} \subset \bar{E}_k$. The leftmost of these $\bar{T}_{q,s}$ is chosen for expansion and becomes $\bar{T}'_{q,s}$. Since $\bar{E}_{k-1} \subseteq \bar{T}_{q,s} \subset \bar{E}_k$, by the inductive hypothesis, after expansion $\bar{T}'_{q,s} \subseteq \bar{E}_k$; therefore, $\bar{T}' \subseteq \bar{E}_{k+1}$.

□

To study the second stage, the following two lemmas are presented.

Lemma 4.7. If $\bar{T}$ is such that $\bar{E}_j \subseteq \bar{T} \subset \bar{F}_j$, then the IncreaseRoot strategy of CNEW yields $\bar{T}'$ such that $\bar{E}_j \subseteq \bar{T} \subset \bar{T}' \subseteq \bar{F}_j$.

Proof: The proof uses induction on j. For brevity, the base step (j=1) is omitted. Assume that the lemma is true for j=1,...,k. It is shown that the lemma is true for j=k+1. Show that given any tree $\bar{T}$ such that

$$\bar{E}_{k+1} \subseteq \bar{T} \subset \bar{F}_{k+1}, \tag{4.33}$$

CNEW expands $\bar{T}$ to yield $\bar{T}'$ such that

$$\bar{E}_{k+1} \subseteq \bar{T} \subset \bar{T}' \subseteq \bar{F}_{k+1}. \tag{4.34}$$

For a tree $\bar{T}$ satisfying (4.33), $\bar{T}$ must be the tree shown in Figure 4.1, where

$$\bar{E}_k \subset \bar{T}_{p,r} \subseteq \bar{F}_k, \text{ for all } p,r\ 1 \le p,r \le w, \tag{4.35}$$

$$\text{and } \bar{E}_k \subseteq \bar{T}_{q,s} \subset \bar{F}_k, \text{ for some } q, s,\ 1 \le q, s \le w. \tag{4.36}$$

If $\bar{T}_{p,r} = \bar{F}_k$, Lemma 3.28 implies that $\uparrow(T_{p,r}, +\infty) = w^k$. When $\bar{T}_{q,s} \subset \bar{F}_k$, Lemma 4.4 implies that $w^{k-1} \le \uparrow(T_{q,s}, +\infty) < w^k$. Therefore, in the first step of consequence in IncreaseRoot, Step I3', M is the set of nodes B_i which have one or more children $\bar{T}_{i,s} \subset \bar{F}_k$. The leftmost of these is selected; denote this to be node B_q. The resulting subsequent call to IncreaseRoot proceeds without effect to Step I4. Now, M is the child(ren) $\bar{T}_{q,s}$ of B_s such that $\bar{T}_{q,s} \subset \bar{F}_k$. The leftmost of these $\bar{T}_{q,s}$ is chosen for expansion and becomes

$\bar{T}'_{q,s}$. Since $\bar{E}_k \subset \bar{T}_{q,s} \subset \bar{F}_k$, by the inductive hypothesis, after expansion $\bar{T}'_{q,s} \subseteq \bar{F}_k$; therefore, $\bar{T}' \subseteq \bar{F}_{k+1}$.

□

Lemma 4.8. If $\bar{T}$ is such that $\bar{F}_{j-1} \subseteq \bar{T} \subset \bar{F}_j$, then the IncreaseRoot strategy in CNEW yields $\bar{T}'$ such that $\bar{F}_{j-1} \subseteq \bar{T} \subset \bar{T}' \subseteq \bar{F}_j$.

Proof: The proof uses induction on j. For brevity, the base step (j=1) is omitted. Assume that the lemma is true for j=1,...,k. It is shown that the lemma is true for j=k+1. Show that given any tree $\bar{T}$ such that

$$\bar{F}_k \subseteq \bar{T} \subset \bar{F}_{k+1}, \tag{4.37}$$

CNEW expands $\bar{T}$ to yield $\bar{T}'$ such that

$$\bar{F}_k \subseteq \bar{T} \subset \bar{T}' \subseteq \bar{F}_{k+1}. \tag{4.38}$$

For a tree $\bar{T}'$ satisfying (4.37), $\bar{T}$ must be the tree shown in Figure 4.1, where

$$\bar{F}_{k-1} \subseteq \bar{T}_{p,r} \subseteq \bar{F}_k, \text{ for all } p,r\ 1 \le p,r \le w, \tag{4.39}$$

$$\text{and } \bar{F}_{k-1} \subseteq \bar{T}_{q,s} \subset \bar{F}_k, \text{ for some } q, s,\ 1 \le q, s \le w. \tag{4.40}$$

If $\bar{T}_{p,r} = \bar{F}_k$, Lemma 3.28 implies that $\uparrow(T_{p,r},+\infty) = w^k$. When $\bar{T}_{q,s} \subset \bar{F}_k$, Lemma 4.5 gives that $w^{k-1} \le \uparrow(T_{q,s},+\infty) < w^k$. Therefore, in the first step of consequence in IncreaseRoot, Step I3', M is the set of nodes B_i which have one or more children $\bar{T}_{i,s} \subset \bar{F}_k$. The leftmost of these is selected; denote this to be node B_q. The resulting subsequent call to IncreaseRoot proceeds without effect to Step I4. Now, M is the child(ren) $\bar{T}_{q,s}$ of B_q such that $\bar{T}_{q,s} \subset \bar{F}_k$. The leftmost of these $\bar{T}_{q,s}$ is chosen for expansion and becomes $\bar{T}'_{q,s}$. Since $\bar{F}_{k-1} \subseteq \bar{T}_{q,s} \subset \bar{F}_k$, after expansion $\bar{T}'_{q,s} \subseteq \bar{F}_k$, by the inductive hypothesis; therefore, $\bar{T}' \subseteq \bar{F}_{k+1}$.

□

Now that the preliminary results have been established, the highlight of this chapter is presented.

Theorem 4.9. Given CT, let $i = \lceil \log_w CT \rceil$. Then, every convergent tree produced by CNEW contains the $\bar{F}_{i-1}$-tree. Furthermore, if the static evaluation function is constant, then the convergent tree produced by CNEW is contained within $\bar{F}_i$.

Proof: While $-\infty$ remains in the range of likely values, whenever CNEW must select a strategy (Step M3), the DecreaseRoot strategy is chosen. The elimination of $-\infty$ will be denoted **Stage I** of CNEW.

Stage I: It is shown that in this stage, CNEW successively produces the trees $\bar{E}_1, \bar{E}_2, \cdots$ until $-\infty$ has been eliminated from the range of likely values.

Initially, CNEW in its first call to DecreaseRoot expands the root node producing (in Step D2) the $\bar{E}_1$-tree. For $i = 1$, $CT \le w$ and Stage I terminates when $\bar{E}_1$ is produced.

For $i > 1$, the range of likely values remains at $[-\infty, +\infty]$. Using Lemma 4.6, clearly CNEW expands $\bar{E}_1$ into $\bar{E}_2$ (in a finite number of expansions), $\bar{E}_2$ into $\bar{E}_3$, ... , and so on as long as it selects the DecreaseRoot strategy in Step M3. Using Lemma 4.3, since $i = \lceil \log_w CT \rceil$, $-\infty$ is eliminated from the range of likely values when the algorithm has built some tree $\bar{T}$ such that

$$\bar{E}_{i-1} \subset \bar{T} \subseteq \bar{E}_i. \tag{4.41}$$

The elimination of $+\infty$ will be denoted **Stage II** of CNEW. Having removed $-\infty$ from the range of likely values by building a tree $\bar{T}$ satisfying (4.41), CNEW next removes $+\infty$ from the range of likely values in Stage II.

Stage II. Whenever CNEW must select a strategy (step M3), the IncreaseRoot strategy is chosen. It is shown that in Stage II, CNEW continues to expand the tree $\bar{T}$ built in Stage I (i. e., the tree $\bar{T}$ satisfying (4.41)), yielding next a tree containing $\bar{F}_{i-1}$. At this point, $+\infty$ is still in the range of likely values. It is shown that further expansions by CNEW (of the $\bar{F}_{i-1}$-tree) yield trees $\bar{T}$ satisfying $\bar{F}_{i-1} \subset \bar{T} \subseteq \bar{F}_i$. For one of these tree $\bar{T}$, $+\infty$ will be removed from the range of likely values. Also, it is shown that if the static evaluation function is constant, then the convergent tree is contained within $\bar{F}_i$.

For a tree $\bar{T}$ satisfying (4.41), $\bar{T}$ must be the tree shown in Figure 4.1 where

$$\bar{E}_{i-2} \subseteq \bar{T}_{p,r} \subseteq \bar{E}_{i-1}, \text{ for all } p,r\ 1 \le p,r \le w. \tag{4.42}$$

From (4.42), clearly either

$$\bar{E}_{i-2} \subseteq \bar{T}_{p,r} \subset \bar{F}_{i-2}, \tag{4.43}$$

or

$$\bar{F}_{i-2} \subseteq \bar{T}_{p,r} \subseteq \bar{E}_{i-1}. \tag{4.44}$$

If (4.43) is true, then Lemma 4.4 implies that

$$w^{i-3} \leq \uparrow(T_{p,r}, +\infty) < w^{i-2}. \tag{4.45}$$

If (4.44) is true, then Lemma 4.5 implies that

$$w^{i-2} \leq \uparrow(T_{p,r}, +\infty) < w^{i-1}. \tag{4.46}$$

If there are any trees $\bar{T}_{p,r}$ satisfying (4.43), then $+\infty$ remains in the range of likely values and in Step I3, some tree $\bar{T}_{p,r}$ satisfying (4.43) (and not (4.44)) is selected for expansion. According to Lemma 4.7, this expansion yields a tree $\bar{T}'_{p,r}$ satisfying

$$\bar{E}_{i-2} \subseteq \bar{T}_{p,r} \subset \bar{T}'_{p,r} \subseteq \bar{F}_{i-2}. \tag{4.47}$$

That is, as long as there is any subtree $\bar{T}_{p,r}$ satisfying (4.43), CNEW expands one such subtree. This expansion yields a subtree $\bar{T}'_{p,r}$ satisfying either (4.43) or (4.44). Since there exist only a finite number of subtrees satisfying (4.43), it follows that after a finite number of expansions, CNEW yields a tree $\bar{T}$ for which its subtrees $\bar{T}_{p,r}$ all satisfy (4.44). At this instance, the tree $\bar{T}$ satisfies

$$\bar{F}_{i-1} \subseteq \bar{T} \subseteq \bar{E}_i. \tag{4.48}$$

This means that every convergent tree produced by CNEW contains the $\bar{F}_{i-1}$-tree.

In a way similar to Lemma 3.28, it may be shown that

$$\uparrow(E_i, +\infty) = w^{i-1}. \tag{4.49}$$

Thus, when $\bar{T}$ satisfies (4.48), (4.49) implies that $+\infty$ is still in the range of likely values. Also, (4.48) implies that

$$\bar{F}_{i-1} \subseteq \bar{T} \subseteq \bar{F}_i. \tag{4.50}$$

According to Lemma 4.8, if the IncreaseRoot strategy of CNEW is applied a finite number of times, $\bar{T} = \bar{F}_i$. Corollary 3.29 implies that $\bar{F}_i$ converges for a con-

stant static evaluation function. Thus, for a constant static evaluation function, the convergent tree produced by CNEW is contained within $\bar{F}_i$.

□

Corollary 4.10: If the static evaluation function is constant, then the number of nodes in the convergent tree produced by CNEW grows quadratically with CT.

Proof: According to Theorem 4.9, for a constant static evaluation function, CNEW produces a convergent tree contained within $\bar{F}_i$. The proof, therefore, follows directly from the proof of Corollary 3.36.

□

Corollary 4.11: For an arbitrary static evaluation function, the number of nodes expanded by CNEW in eliminating $+\infty$ and $-\infty$ from the range of likely values grows quadratically with CT.

Proof: Given CT, let $i = \left\lceil \log_w CT \right\rceil$. From Theorem 4.9, the tree $\bar{T}$ obtained by CNEW contains the $\bar{F}_{i-1}$-tree. For a given, arbitrary static evaluation function, if either $-\infty$ or $+\infty$ is still in the range of likely values, the algorithm continues by expanding $\bar{T}$. As in the proof of Theorem 4.9, CNEW expands a subtree $\bar{T}_{k,l} \subset \bar{F}_{i-2}$ of $\bar{T}$ and the resulting tree $\bar{T}'$ satisfies $\bar{T}' \subseteq \bar{F}_i$. According to (3.79) and (3.80),

$$\uparrow(F_i, +\infty) = w^i$$

and

$$\downarrow(F_i, -\infty) = w^i$$

Thus, after a finite number of expansions, $+\infty$ and $-\infty$ are eliminated from the range of likely values. The result follows from (3.92).

□

The results of Theorem 4.9 are illustrated by Tables 4.1 and 4.2 below. The results of the tables have been verified through experiments performed by modifying an implementation of McAllester's algorithm [Klin89] to change it into CNEW.

For a fixed static evaluation function, the improvement of CNEW over McAllester's algorithm may be seen by comparing Tables 4.1 and 4.2 with Tables 3.1 and 3.2. For example, for CT = 100 and w = 20, McAllester's algorithm produces a tree with 1344444444221 nodes; CNEW produces a tree with only 11111 nodes!

	w = 2		w = 3		w = 4		w = 5	
CT	d	N_{CT}	d	N_{CT}	d	N_{CT}	d	N_{CT}
2	2	7	2	13	2	21	2	31
3	4	21	2	13	2	21	2	31
4	4	31	4	55	2	21	2	31
5	6	61	4	55	4	113	2	31
6	6	87	4	91	4	113	4	201
7	6	109	4	91	4	113	4	201
8	6	127	4	121	4	197	4	201
9	8	189	4	121	4	197	4	201
10	8	247	6	253	4	197	4	361

Table 4.1. Depth and number of nodes in CNEW tree.

	w = 10		w = 20		w = 30		w = 40	
CT	d	N_{CT}	d	N_{CT}	d	N_{CT}	d	N_{CT}
10	2	111	2	421	2	931	2	1641
20	4	2471	2	421	2	931	2	1641
30	4	3621	4	9201	2	931	2	1641
40	4	4751	4	17941	4	29701	2	1641
50	4	5861	4	17941	4	29701	4	68801
60	4	6951	4	26641	4	58411	4	68801
70	4	8021	4	26641	4	58411	4	68801
80	4	9071	4	35301	4	58411	4	135881
90	4	10101	4	35301	4	87061	4	135881
100	4	11111	4	43921	4	87061	4	135881

Table 4.2. Depth and number of nodes in CNEW tree.

4.3. The Success of CNEW

It is appropriate now to discuss the significance of CNEW. Many questions arise; the most general one asks whether the new algorithm will be an improvement over McAllester's algorithm.

For a constant static evaluation function, CNEW is significantly better than McAllester's algorithm (compare Corollary 3.27 with Corollary 4.10; also, compare Tables 4.1 and 4.2 with Tables 3.1 and 3.2). Both algorithms always converge (see Corollary 3.20 and Corollary 4.10). However, the number of nodes expanded by CNEW grows quadratically with CT, whereas the number of nodes expanded by McAllester's algorithm grows exponentially with CT.

For an arbitrary static evaluation function, the number of nodes expanded by CNEW grows quadratically with CT while eliminating $+\infty$ and $-\infty$ from the range of likely values. Recall that Corollary 3.20 showed that McAllester's algorithm always built a $\bar{C}_m$-tree while making the range of likely values finite. That is, for an arbitrary static evaluation function, McAllester's algorithm expands an exponential number of nodes in the same, predictable manner, regardless of the static evaluation function. Thus, CNEW looks promising. Empirical tests on the modified algorithm should confirm its usefulness.

At this time, it is not possible to state unequivocally that CNEW expands fewer nodes than McAllester's algorithm for an arbitrary static evaluation function. It is possible that the initial improvement (while eliminating $+\infty$ and $-\infty$ from the range of likely values) is counteracted in the later stages of the algorithm; perhaps the CNEW algorithm always produces a $\bar{C}_m$-tree anyway, or another tree with even more nodes than the McAllester tree. Also, there is no guarantee that CNEW will converge, just as there was no guarantee that McAllester's algorithm would converge.

The nurturing of a baby in its first years has a major impact on its subsequent success in life. Similarly, it is intuitively felt that the gain in the initial stages of the CNEW (when making the range of likely values finite) will carry over into the later stages of the new algorithm. It is thought that the CNEW algorithm is a significant improvement over the McAllester algorithm, and it is hoped that work done in the future will confirm the utility of the new algorithm, either empirically or theoretically.

It was shown in Theorem 3.35 that for a constant static evaluation function when $CT \geq w^3$, there existed a convergent full tree with fewer nodes than a uniform-McAllester tree. Under the assumptions made throughout this thesis, that the branching factor w is constant throughout and that every child of an expanded node is evaluated, it is conjectured that for *any* CT, CNEW produces the tree with the fewest nodes which removes $-\infty$ and $+\infty$ from the range of likely values. (Recall Conjecture 3.39).

Chapter 5

Conclusions

5.1. Accomplishments of this Thesis

This thesis investigates the behavior of McAllester's algorithm. For a given CT and a fixed static evaluation function, this thesis derives and proves formulaes for calculating the number of nodes expanded by the algorithm as well as the depth of the tree produced. One interesting result is that the growth factor decreases as w increases. Therefore, except for small CT, for a constant static evaluation function and a given CT, as w increases, the number of nodes expanded decreases.

A main achievement of this thesis is a lower bound analysis of McAllester's algorithm. It is shown that the lower bound occurs when all the leaf nodes have the same static value. Lower bound analysis shows that McAllester's algorithm always expands at least a $\bar{C}_m$-tree, where $m = \lceil \log_w CT \rceil$. If all the leaf nodes have the same static value, then this tree is convergent. The analysis shows, as expected, that the number of nodes expanded in producing a $\bar{C}_m$-tree grows exponentially with CT.

A much stronger result is obtained. It is shown that for an arbitrary static evaluation function McAllester's algorithm always starts by growing a particular tree of a predictable shape. This tree is the same $\bar{C}_m$-tree. It is produced while McAllester's algorithm eliminates $-\infty$ and $+\infty$ from the range of likely values. Thus, eliminating $-\infty$ and $+\infty$ from the range of likely values is not done in a data-dependent way; the static evaluation function does not guide the search in this stage of the algorithm. Although McAllester's algorithm is not a strict depth-first or breadth-first search, it is a brute-force search while it eliminates $-\infty$ and $+\infty$ from the range of likely values! Since the number of nodes expanded in producing a $\bar{C}_m$-tree grows exponentially with CT, this conclusion is important.

Another major contribution of this thesis is an enhancement of McAllester's algorithm. It seemed intuitive that work would be saved if, when trying to make the root attain a particular value, one expanded the node requiring the fewest conspirators to attain that value. While studying McAllester's algorithm, it

was observed in Sections D3 and I3 that this principle was not followed (McAllester simply expands the left-most node). Independent of the static evaluation function, McAllester's algorithm may expand some nodes only to depth two while expanding other nodes much further. That is, McAllester's algorithm is prejudiced towards the expansion of certain nodes, without any justification relating to the static evaluation function. This observation led to the creation of an improved algorithm, CNEW. A lower bound analysis of this algorithm (i. e. all the leaves have the same static value) showed that the number of nodes expanded by CNEW while eliminating $-\infty$ and $+\infty$ from the range of likely values grows only quadratically with CT.

5.2. Suggestions for Further Work

This thesis obtains a lower bound analysis of McAllester's algorithm because it was decided that analysis of the algorithm in general (with an arbitrary static evaluation function) would be extremely difficult. Perhaps someone will solve this problem. It is more likely, however, that someone will consider answering some (but not all) of the following issues that arise for McAllester's algorithm with an arbitrary static evaluation function. Can one state, *a priori*, what type of static evaluation function will lead to a convergent tree? One method of studying the problem might be to take a tree that is convergent for a given CT and a fixed static evaluation function and perturb it by changing the static values of some leaves; is the tree still convergent?

Another question that hasn't been addressed is this: is it a necessary condition for convergence that some or all subtrees have a constant static evaluation function? Considering questions such as these relating to McAllester's algorithm might lead to the conclusion that McAllester's algorithm will only converge if the static evaluation function is "fairly stable", either locally or globally. It seems probable that this is true; it remains a research question to define the term "fairly stable" in a more precise manner.

The writing of this thesis has led to many ideas for enhancements of McAllester's algorithm that have not been analyzed or implemented. Some of them are presented in the following discussion as suggestions for future research.

Both McAllester's algorithm and CNEW obtain minimax values for all possible moves. Each move is studied until a desired degree of accuracy (determined by the CT parameter) has been achieved for the minimax value. If there are several moves with the highest minimax value, the two algorithms have no way to distinguish between them. Rather than choosing one move at random, it is desirable to have a scientific method of selecting a move. One possible method would be to continue the algorithm (McAllester's or CNEW) *on the tied (best) moves only* with a higher CT. This may be one way to improve McAllester's algorithm or CNEW.

Another improvement has been considered. Since McAllester's algorithm always starts by building a $\overline{C}_m$-tree and CNEW always begins with an $\overline{F}_{i-1}$-tree, simply commence by building that tree! Klingbeil [Klin89] built a full tree of depth 2; starting with an $\overline{F}_{i-1}$-tree is a major extension of his idea that should have a much greater impact.

References

[AHU83]

A. Aho, H Hopcraft, J. Ullman, "Data Structures and Algorithms", pp. 105, Addison Wesley Publishing Company, London, 1983.

[ACH88]

T. Anantharaman, M. Campbell, F. H Hsu, "Singular Extensions: Adding Selectivity to Brute-Force Searching", Journal of the International Computer Chess Association #11, 1988, pp. 135-143.

[Baud78]

G. Baudet, "On the Branching Factor of the Alpha-Beta Pruning Algorithm", Artificial Intelligence #10, 1978, pp. 173-199.

[Berl73]

H. Berliner, "Some Necessary Conditions for a Master Chess Program", Procs. 3rd Int. Joint Conf. on Art. Intell., Stanford, 1973, pp. 77-85.

[Berl79]

H. Berliner, "The B^* Tree Search Algorithm: A Best-First Proof Procedure", Artificial Intelligence #12, 1979, pp. 23-40.

[Camp83]

M. Campbell, "Algorithms for the Parallel Search of Game Trees", Technical Report TR 81-8, Department of Computing Science, University of Alberta, 1981.

[Klin89]

N. Klingbeil, "Conspiracy Numbers", Master's Thesis, University of Alberta, 1989.

[KS88]

N. Klingbeil and J. Schaeffer, "Search Strategies for Conspiracy Numbers", Proceedings of the 7th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, 1988, pp. 133-139.

[KM75]

D. E. Knuth and R. W. Moore, "An Analysis of Alpha-Beta Pruning", Artificial Intelligence #6, 1975, pp. 293-326.

[Li89]

L. Li, "Probabilistic Analysis of Search", Ph. D. Thesis, University of Alberta, 1989.

[Mars87]

T. Marsland, "Searching for Chess", Technical Report TR 87-6, Department of Computing Science, University of Alberta, 1987.

[McAl85]

D. McAllester, "A New Procedure for Growing Mini-Max Trees", Technical Report, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1985.

[McAl88]

D. McAllester, "Conspiracy Numbers for Min-Max Search", Artificial Intelligence #35, 1988, pp. 287-310.

[Nau83]

D. S. Nau, "An Investigation of the Causes of Pathology in Games", Artificial Intelligence #19, 1982, pp. 257-278.

[Nils71]

N. Nilsson, "Problem-Solving Methods in Artificial Intelligence", McGraw-Hill, N.Y., 1971, pp. 137-139.

[Pala85]

A. Palay, "Searching with Probabilities", Pitman, Boston, 1985, pp. 33-34.

[Pear84]

J. Pearl, "Heuristics: Intelligent Search Strategies for Computer Problem Solving", Addison-Wesley, London, 1984.

[Scha89]

J. Schaeffer, "Conspiracy Numbers", Advances in Computer Chess V (Ed. D. Beal), North-Holland, New York, 1989, pp. 199-218.

[SD69]

J. R. Slagle, and J. K. Dixon, "Experiments with some Programs which Search Game Trees", Journal ACM 16:2, 1969, pp. 189-207.

[Shan50]

C. E. Shannon, "Programming a Computer or Playing Chess", Philosophical Magazine (Series 7) 41, 1950, pp. 256-275.

[Stoc79]

G. Stockman, "A Minimax Algorithm Better Than Alpha-Beta?", Artificial Intelligence #12, 1979, pp. 179-196.